Design Document (CS-4500 Fall 2022)

Project Code Blue: Content Recommendation

(Alec Chae, Samuel Gera, Scott Kenner, Aaron Mayer, Liming Xue)

October 24th, 2022

Contents

I Executive Summary

Our project is oriented towards tracking and recommending users content and utilizing cutting edge data mining and machine learning techniques. This will enhance the users experience exploring the web, and provide unbiased recommendations towards a single site. The utilizations of google extensions to both track the user to provide recommendations, and the delivery and readability of RSS feeds will be the main crux of service that will be provided to the user. Machine learning and its application to this project are the main processors of this data to provide the user experience, allowing a long running process. Particularly we will be implementing a version of google's BERT language processing model to accomplish this task. This local utilization of hardware to overcome expensive data centers and decentralizing the processing of data to not only allow security and anonymity, but transparency into these factors insomuch as the tenants of data science allows.

See appendix section A for a brief diagram of the user experience This service is unique and novel to allow the user more power and control over harvested information, and unbiased access to more content since the only requirement to get recommended content is the availability of it on an RSS feed. This is unlike google, reddit, and other aggregators or services that are inherently biased to keep people on their own sites, due to the bulk of the service being done locally, via the machine learning application being done on local hardware, there is extraordinarily little overhead for our own service.

II Background and Technical Requirements

II.1 Background

In the world today there is a sense of big data harvesting ever more information, while the consumer hardware is also exponentially speeding up in terms of data processing capability. What once could be used to run DOOM on your family computer now is almost the same hardware actually recommending you videos on youtube, or webpages on google. Since we can leverage these devices, we can allow you to consume content from webpages, decentralizing the processing power to a user level and allowing more control of privacy and security.

Obviously content recommendations have been around for a while, google launching as a primitive version of this in the late 90s.

II.2 Technical Requirements

1. Mongodb
2. Python
3. Pytorch
4. Flask
5. Javascript
6. Bootstrap
7. Script Injection
8. HTML parsing
9. Local GPU Acceleration and Utilization

II.3 Hardware Requirements

- CPUs can perform the necessary training and recommendations, but take significantly longer
- A Modern GPU Is required to run this software with reasonable performance (Nvidia current test bench)
- Windows (Linux may be an acceptable OS if the dependencies are fine running on it, though support is not promised in this document)

III Requirement Analysis

For hardware you need a relatively modern GPU, arbitrarily GTX 10XX series or newer Windows, local install of mongodb, installation of the various python libraries (flask, cerberus) chromium based browser for extension, modern browser capable of displaying bootstrap style sheets and HTML5.

III.1 System Architecture (Componenets)

Simplified graph below, for a full view of the System Architecture consult figure **9**.

$$Extension -> Flask(Server) -> storage -> processing -> returning$$

III.1.1 Extension

Scrapes the user's live data using the web extension. Uses bootstrap to make the front-end more user-friendly.

III.1.2 Flask Server/ Desktop App

Runs a local server using python code to communicate with chrome extension. The desktop application will be managing this server, mainly for configuration issues and local access to hardware resources such as the GPU. It should be noted this server is run as a local desktop application due to the restraints of the web extensions not allowing. The desktop application will also act as a container for the processes within the processing step.

III.2 Content Recommendation Model

Currently we plan to do content recommendation using cosine similarity of feature vectors generated from the articles. In order to do this well we need to select a set of features that comprehensively describes each article's content and accurately map articles to these features. The current approach is to do some transfer learning on an existing natural language model with a data set containing articles and their corresponding topics. Last semester we used BERT but I may switch it out for XLNet this time. We will of course need a GPU to train this model but hopefully we won't to deploy it. If the deployed model is too computationally intensive to run on a standard computer we may at least attempt to distill the model to mitigate this.

III.2.1 Storage

Stores the JSON Data received from our chrome extension to our database: Particularly we are using mongoDB as a database with a legacy of being used in large machine learning operations, organized using the flask server..

III.2.2 Returning

On Extension UI, confidence intervals and recommendations displayed/scored.

III.3 System Architecture (Interfaces)

III.3.1 Desktop Application

The desktop application serves as a medium for the user to interact w/ and control the training processes that the model has to perform on the user's hardware. It will provide information regarding GPU usage statistics and training progress. It will also allow the user to pause and cancel any local processes. (Flask server/ Model) Mockups for the UI of the desktop application can be seen in figure **8**.

III.3.2 Extension

The extension acts as a medium through which users can manage the privacy features of the HTML scraper that is built into the web extension. It also acts as an interface through which users can receive their content recommendations. Mockups of the extension can be seen in figure **7**.

III.4 Personnel

III.4.1 Alec Chae

Alec was responsible for database system for this project. Alec was able to narrow down and decided that mongoDB would be adequate for our project. So far we have not been utilizing any of the functions that are embedded into MongoDB and in the future Alec is looking to research more on what functions MongoDB provides and utilize any of the functions if necessary. We have been using Flask Server to communicate between our MongoDB and Extensions. During the communication we are using a library called Cerberus to validate the Data that are being sent from the Extension using Flask Server. In the future, I would have to consider more restrictions so that it is only using appropriate data to compute our content based recommendation. Alec has been working on the front-end for chrome extension to make it more user friendly.

III.4.2 Samuel Gera

Sam was responsible for creating a desktop application that worked as a container for Scott and Aarons contributions to the project, as well as help initially design the infrastructure for the content recommendation pipeline with Scott and Aaron.This UI for the desktop application was developed in Kivy, and it served as an idle animation container, containing updates on what the model was currently doing as a background process (training, recommendations, communications etc.). As of CS 4500 Sam is mostly responsible for the UI of the desktop application and it's integration with the Model and Flask Server componenets.

III.4.3 Scott Kenner

Scott worked as the Team coordinator, frequently driving meetings and sprint planning discussions. Scott also largely contributed to the design of the entire infrastructure our current working prototype is comprised of. Furthermore, Scott worked on the HTML parser that used beautiful soup and other packages in order to parse content from numerous RSS feeds to generate recent content recommendations based on what was retrieved.

III.4.4 Aaron Mayer

Aaron worked as the main engineer for our Topic Modeler. He primarily developed and implemented everything related to generating content recommendations, from a data science perspective, using pytorch and an existent data set to train the topic modeler on. This was used later in tandem with Sam's and Scott's contributions in order to generate content recommendations.

III.4.5 Liming(Max) Xue

Max Developed the chrome extension in Javascript. It was primarily responsible for scraping data related to a users actions on a webpage. HTML documents and page activity was retrieved, in order for Alec to be able to store those values for the model to use later. A UI that also displayed scores, and allowed privacy options (Disabling activity recording) was also provided.

NOTE: Most of this section was filled out by Sam, if contributions seem uneven, it may be because I did not describe them sufficiently! (I think we did a great job as a team this semester)

III.5 System Features

III.5.1 Rank 1

B1 Data processing

B2 Basic metrics

B3 Extension

B4 Content scraper for recommendations

III.5.2 Rank 2

P1 Optimizations on the data processing model to improve efficacy of results after metrics are taken into account.

P2 Installation process for application for easy setup

P3 Gpu usage limiting to allow user smooth experience when application is doing heavy data processing. As well as GPU usage statistics interfaces

P4 Desktop Extension Integration to smooth unified user experience

P5 A system of cached recommendations for quick user recommend versus on demand processing.

P6 Generalized UI Polish for Application

III.5.3 Rank 3

W1 Anonymous data sharing for additional metrics (recommendation cross referencing see figure 5)

W2 additional metric collection outside of browser eg. times playing games, watching movies,mouse movements)

W3 Additional and complex metrics, (see yellow and red boxes in figure 3)

IV Software Engineering Tools/Techniques

IV.1 Development Planning/Organization Tools

- Scrum
- Agile Pointing
- Story Focused Feature Creation
- Bi-Daily Stand ups
- GitHub Versioning Branching

https://www.overleaf.com/project/6201ef4fc474edcd153c78d7

IV.2 Development Overview

Particularly we utilize agile development to get feature-first focused work out especially in the second half of the class we want to focus on getting key features out, and not focusing on "pork" that bloats the project unnecessarily. Particularly through defining the scope of work and having user based practical discussions about the work needed to implement a feature, and the priority of that feature to the efficacy to the user is key. Key concept of practicality avoids scenarios such as "adding these features breaks the entire program, and we can't fix it in time without shifting work from another feature" It's obvious that such a feature/ work story needs to be managed carefully. Avoiding mismanagement of complexity is key since the work on the project must converge by December.

Additionally, we want to continue to utilize a more stabilized development environment, currently the installation of the project to run requires multiple steps, and the simplification of the project to run as smoothly as possible on people computers needs to be streamlined far further. It already is a two step process, being an extension and application, so the complexity of install is something to keep an eye on and reducing the variable-ness of development installation and builds would go a long way.

V Timeline

Fall Semester Initialization		
DD1	9/9	Scott
Pre Fall Break Milestones		.
B1	10/16	Aaron
B3	10/16	Max
P4	10/16	Sam
P5	10/16	Alec/Scott
Post Fall Break Milestones		
UG1	10/21	Scott
DD2	10/21	Scott
Beta Demo	10/26	Scott
P1	11/27	Aaron
P2	11/27	Alec/Scott
P3	11/27	Sam
P6	11/27	Max/Alec/Sam
Polish and Presentation Goals		
UG2	12/2	Scott
DD3	12/2	Scott
Demo	Demo Day	All
Unfeasable/Unassigned		
W1	Unplanned	Unassigned
W2	Unplanned	Unassigned
W3	Unplanned	Unassigned

VI Appendix A: UI Sketches and UX flow

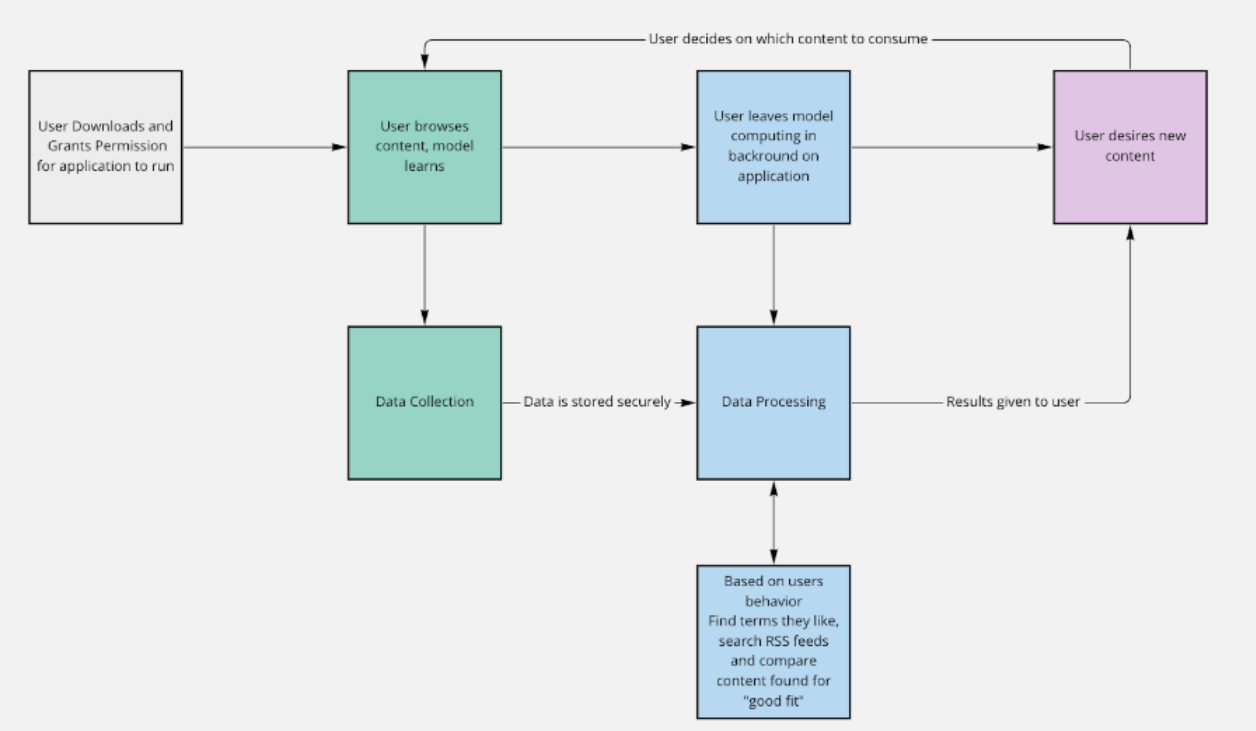

Figure 1: User experience within the application

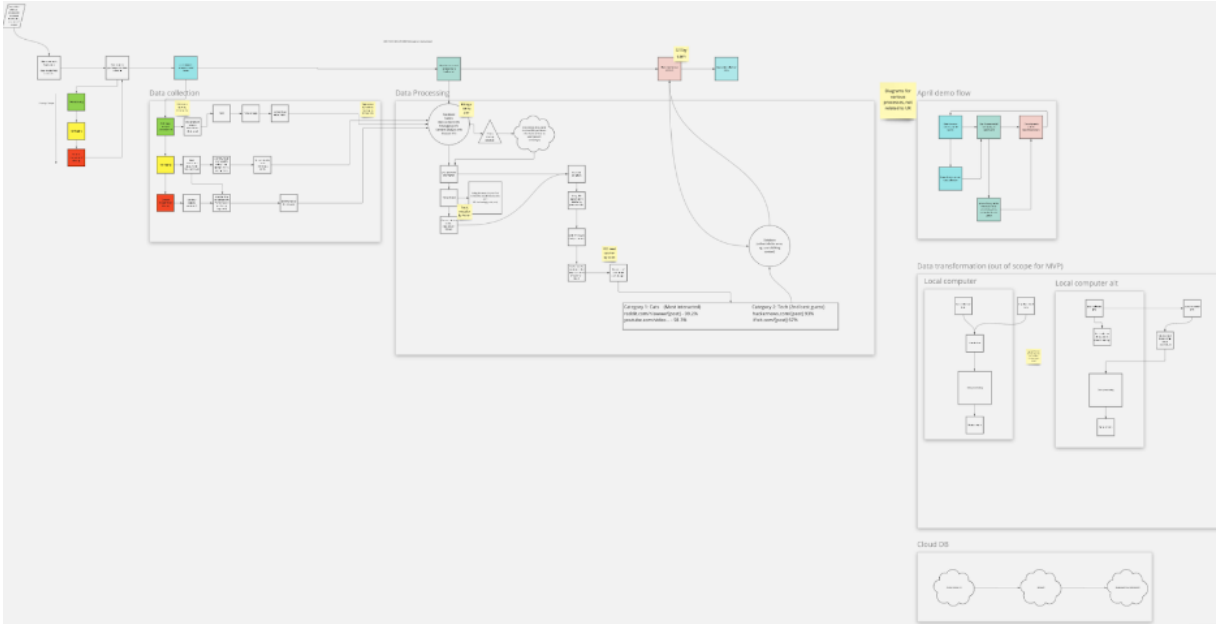

Figure 2: zoomed out overview of technical specifications for the project with the main user experience over the top of the technical specifications (Not supposed to be readable, just displaying the stage of our understanding)

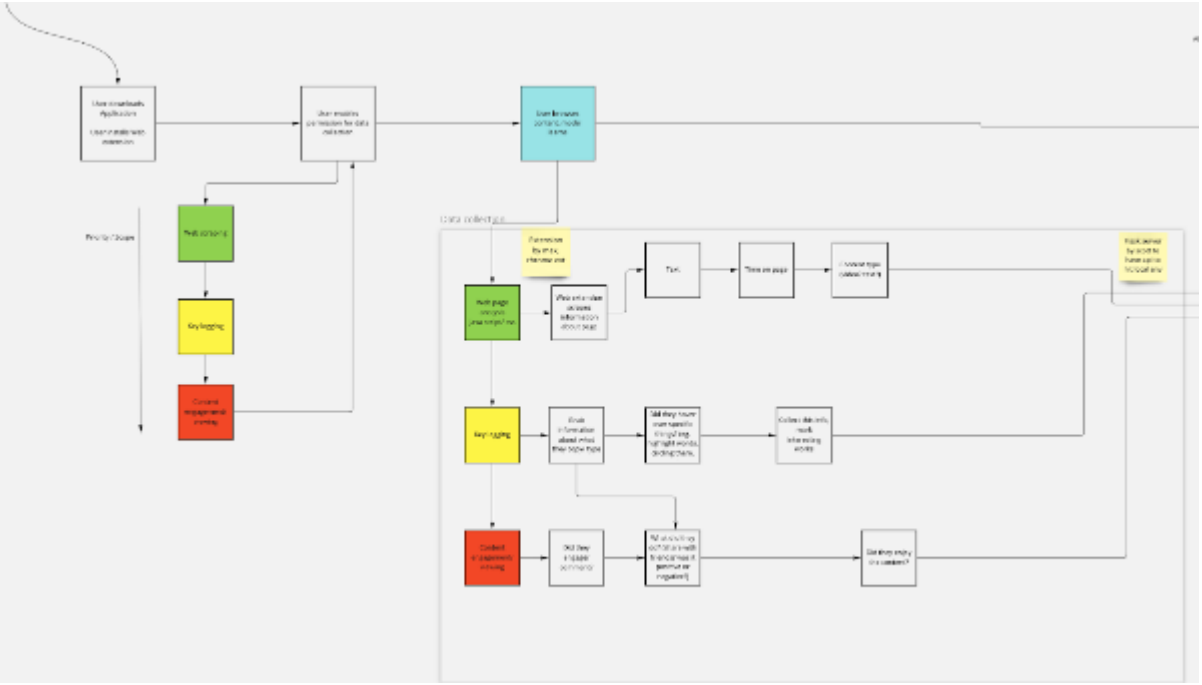

Figure 3: data collection and initial metric collections in the web browser via the chrome extension

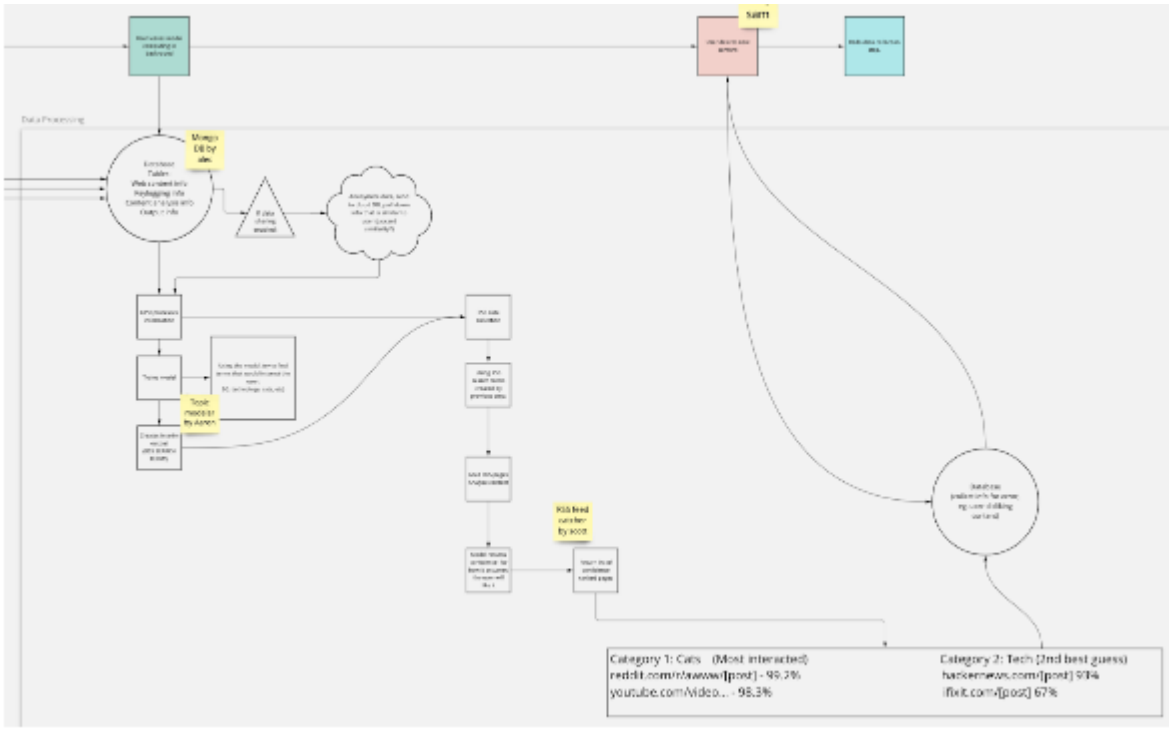

Figure 4: Data processing and machine learning pipeline for the project, returning recommendations

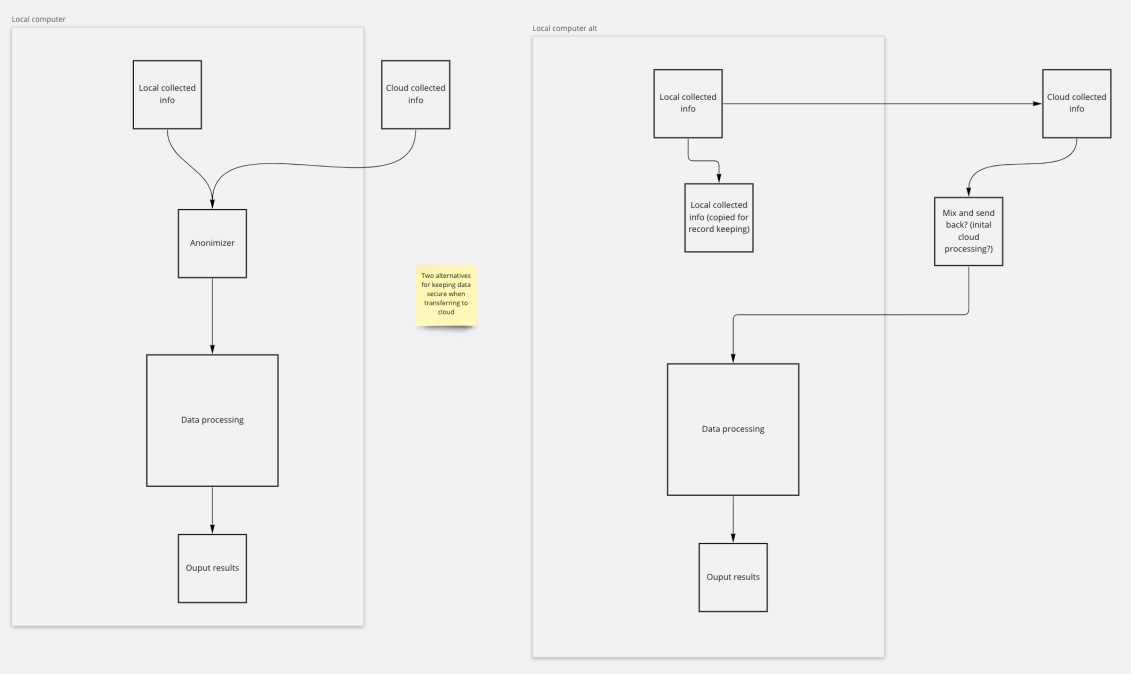
Local computer
Local collected info
Cloud collected info
Anonimizer
Data processing
Ouput results
Two alternatives for keeping data secure when transferring to cloud
Local computer alt
Local collected info
Cloud collected info
Local collected info (copied for record keeping)
Mix and send back? (inital cloud processing?)
Data processing
Ouput results

Figure 5:

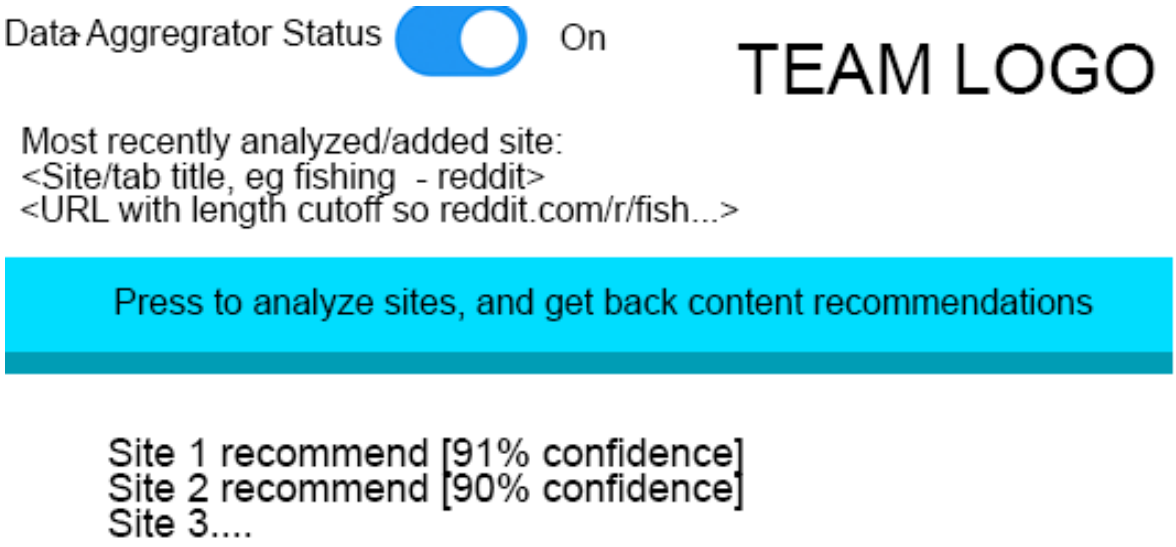

Figure 6: basic chrome extension mock-up depicting features and layout for displaying content recommendations and their confidence intervals

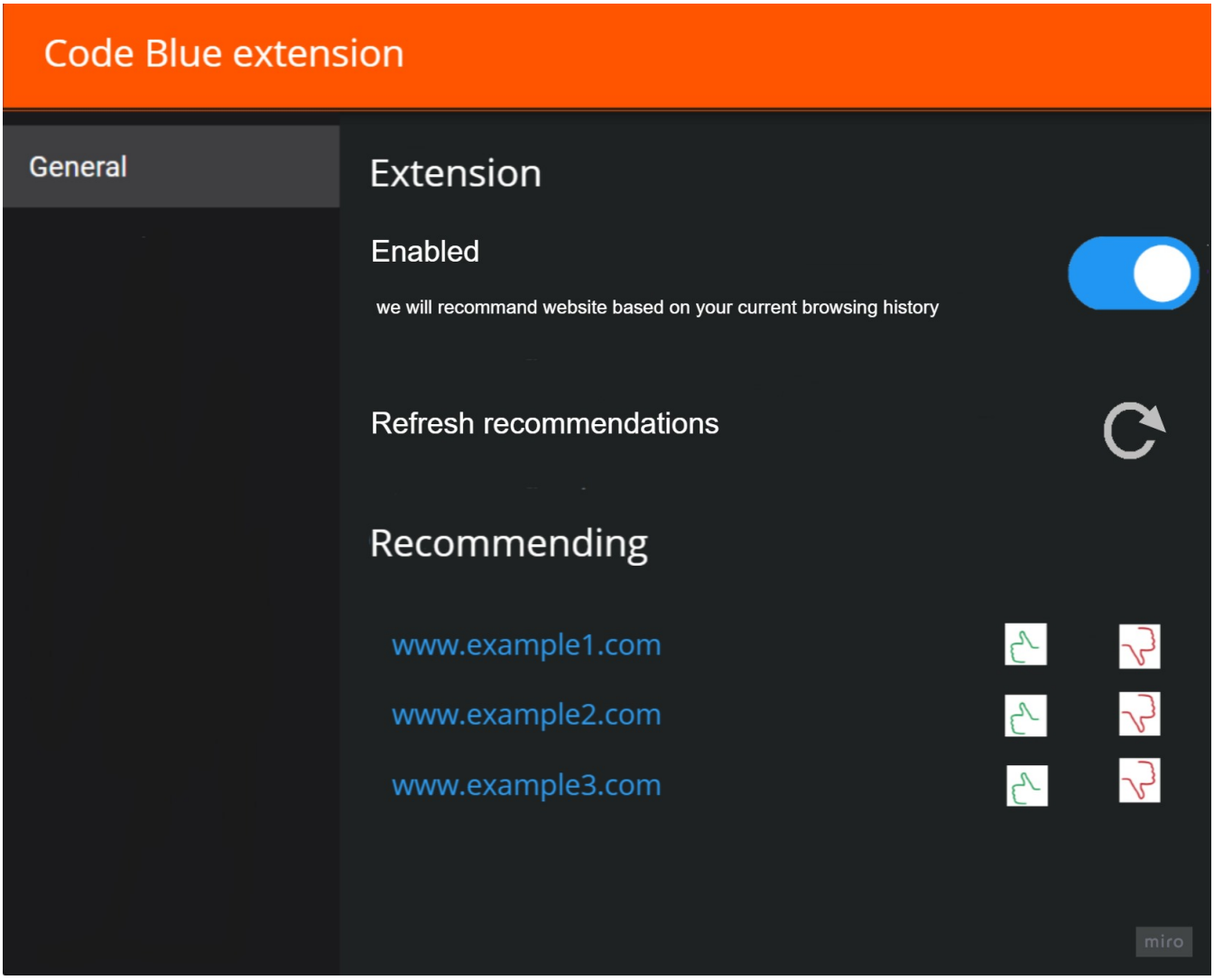

Figure 7: wire-frame of the extension

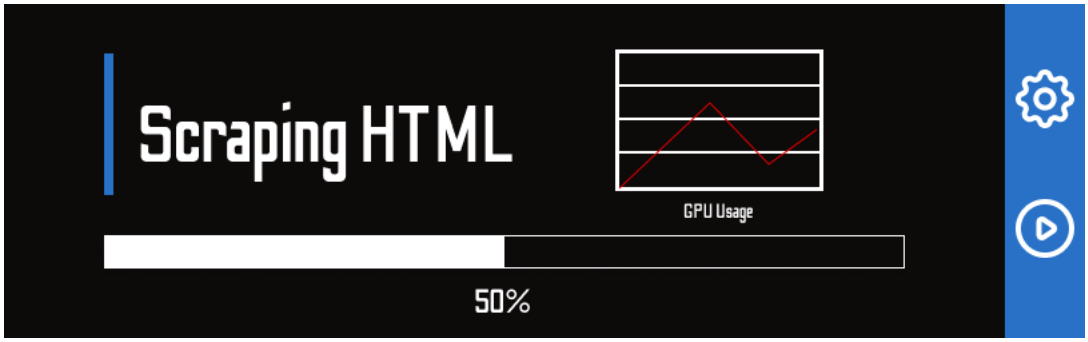

Figure 8: Updated wire-frame for the desktop application (updated 9/9). Aims to give user information on the progress of training processes, and GPU/CPU usage, as well as the possibility to pause usage.

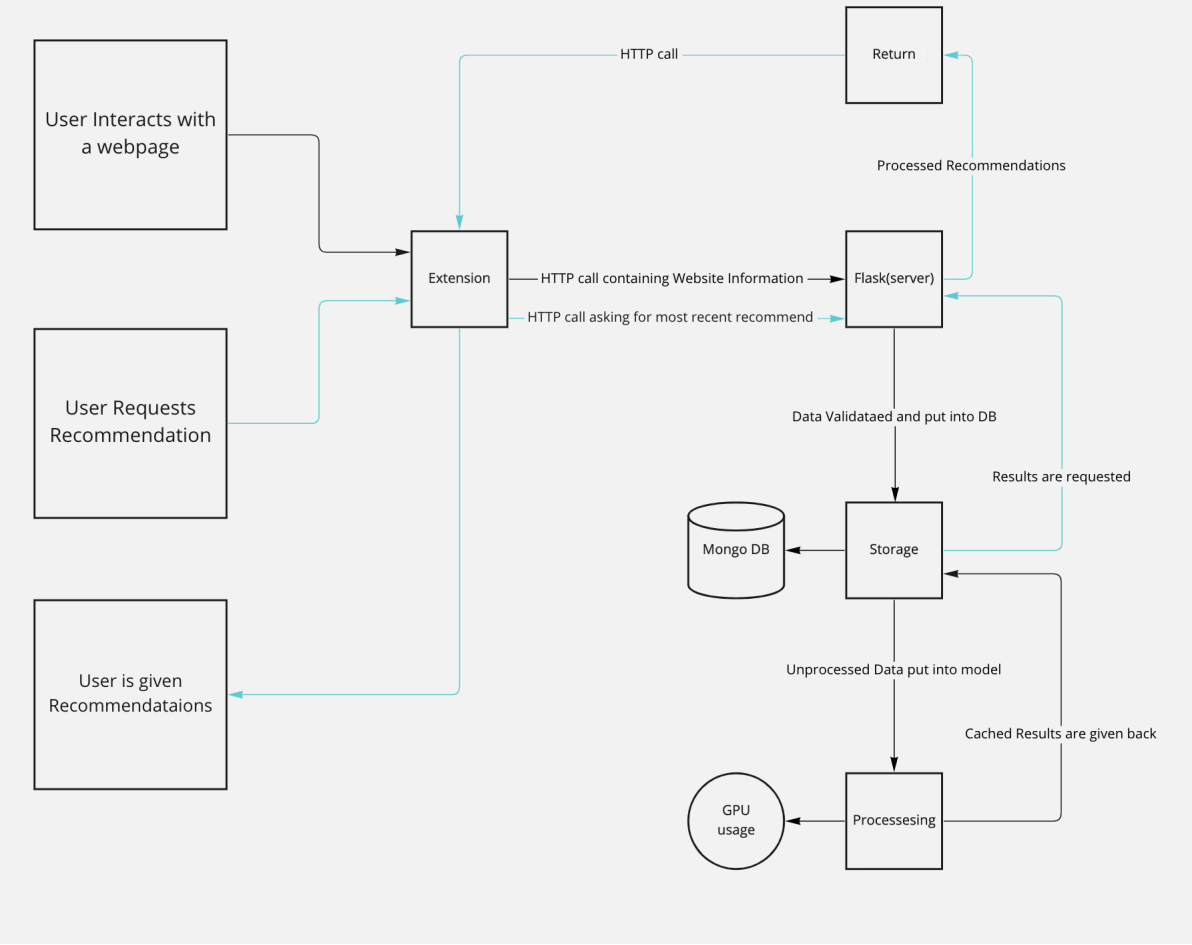

Figure 9: A view of the system architecture, blue lines indicate the second step where the user is requesting information versus the automatic process of data collection when user interacts with a web-page.

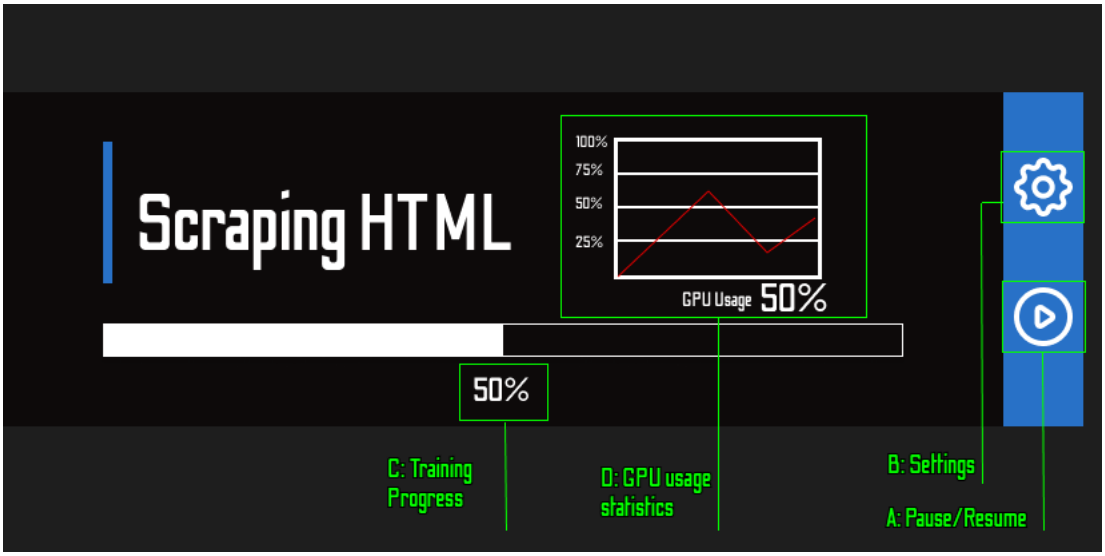

Figure 10: A labeled view of the desktop application wire-frame containing GPU usage statistics, interfaces, and a progression bar.

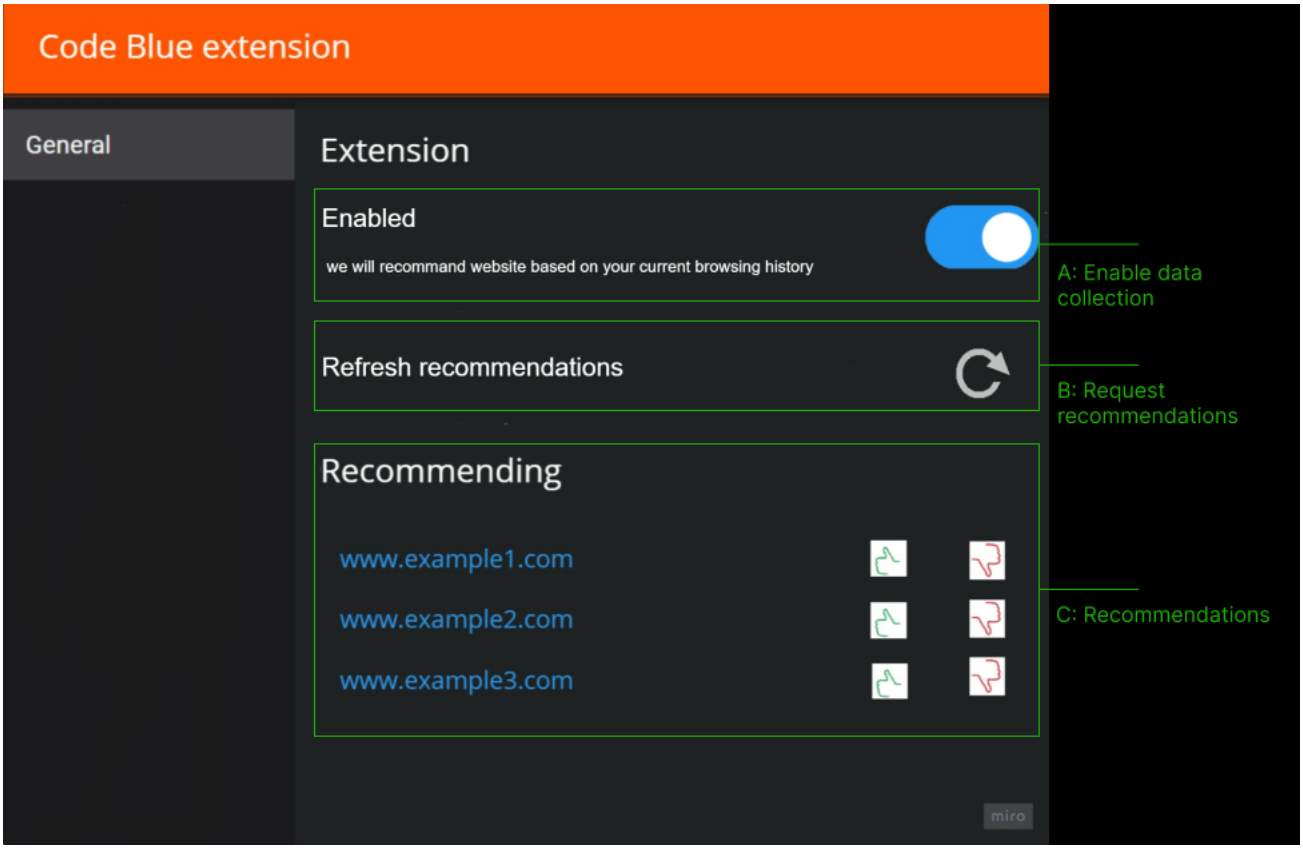

Figure 11: A labeled view of the extension wire-frame containing interfaces for the user to specify data collection, request recommendations, and rate received recommendations

VII Appendix B (Use Cases)

VII.1 Scraping Information

The system/model needs to collect information to recommend content, this is the basis of a lot of our work, in this case as the extension notices you browsing, it will go ahead and pass along valid HTML to the server (some validation such as blocking out website with unwanted content to recommend as directed via the user will be skipped.) Once at the application it will then be process and impressed into the model

VII.2 Recommending Web-pages

The application should after a bit of caching be ready to recommend content, and the request, sent via the blue button in figure 1, will go through to bring in pages for recommendation. There is a bit of thinking and testing to go into the exact parameters of caching for speeding up click to recommend times, along with utilizing pre-impressed pages to sync up every so often.