A SMART CHROME EXTENSION FOR WEB CONTENT ANALYSIS AND DISTRACTION MITIGATION USING ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Zhiyuan Zhu [1], Garret Washburn [2]

[1] University High School, 4771 Campus Dr, lrvine, CA 92612
[2] Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

The temptation to lose oneself in the digital world is more prevalent now in the present state of the internet than ever before. Many internet users often fall into this temptation, voiding their efforts to get any productive work done that they may need to utilize their internet browser. To solve this problem, this paper proposes the ScreenTimeSage Chrome Extension available on the Chrome Web Store [12]. The ScreenTimeSage Chrome Extension seeks to remedy this issue of internet users getting off task but utilizing AI to analyze their specific browsing behavior and analyzing where and how they tend to get off task. Once the extension has analyzed how the user gets off task, it is able to provide customized support and advice on how to help the user stay focused. This extension relies on the Gemini LLM API, as it utilizes prompt engineering and Chrome storage to keep an active record of the user's browsing habits, and classifying how productive the user is on each web page [13]. This extension has been tested for its content classification as well as its back-end server reliability, which is available in this paper. Both experiments are demonstrations of good results from the extension. The ScreenTimeSage Chrome Extension is ultimately a great option for those seeking a way to combat their inclination to get distracted while using their browser, as it provides customized insights on how they individually get distracted and how they can combat this for themselves.

KEYWORDS

Productivity, AI-driven browsing analysis, Distraction, Chrome extension

1. INTRODUCTION

With the rise of the internet and social media, many struggle with controlling their time spent on the "digital world". The overconsumption of the "digital world" often can lead to many harms to users' health, including mental health issues as well as developmental issues. A research study held in the American Economic Review found that 24% of the drastic increase of depressed college students can be attributed to the introduction of Facebook [1]. Individuals who are subject to overusing the internet are increasingly likely to get off task when trying to complete their intended goal during their stay on the internet.

The first methodology examined in this paper is Apple's Screen Time application available on Apple devices including Mac and Iphones [2]. While Apple Screen Time is likely the most

David C. Wyld et al. (Eds): BDAB, SVC, AIBD, ACSTY, NATP – 2025
pp. 93-101, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2024.150108

familiar and notorious methodology for tracking how one spends their time on their mobile device, it is not capable of providing insights on how each application is used or providing feedback. The ScreenTimeSage Extension seeks to provide more in depth insights as well as customized feedback for the individual.

The second methodology compares the program to LifeAt desktop application. Although LifeAt provides a simulated and customizable environment with sound plus features like to-do lists, it ultimately lacks any analysis of the user's screen time usage nor providing any depth insights on user's behavior. In contrast, The Screen Time Sage such features, which we believe will better enable the user to reflect moderate their screen time usage, the purpose of both functions.

Blocksi is another methodology for ensuring the user is on task on their browser, however, it is more in the approach of allowing educators, such as teachers, be able to monitor their current browser activity to ensure they are on task. While this may be productive in a classroom setting, it is not beneficial in helping shape long term habits for the students. Therefore, ScreenTimeSage is a better alternative as it allows for the shaping of habits.

The method proposed within this paper to solve the issue as described above is the Screen Time Sage Chrome Extension, available on the Chrome Web Store [3]. The Screen Time Sage Chrome Extension is an extension focused on monitoring the users internet browsing behavior and providing useful analysis on their habits and tendencies to get distracted. Not only that, but the Screen Time Sage extension can provide the user with helpful feedback and tailored instruction to avoid getting distracted for that specific individual. The Screen Time Sage Chrome Extension provides the user with a fully interactive experience, designed to give quality feedback in order to help them improve on their ability to keep engaged in their work. Opposed to other methodologies users tend to use as a solution, Screen Time Sage is designed to work with the user - not against the user's habits to get distracted. The unfortunately unique strategy Screen Time Sage takes to solving this issue aids the user in keeping focus more effectively. This is because it provides in depth analysis on the user's interaction with their browser, enabling behavior analysis - which is incredibly useful for providing advice on how to stay engaged in completing tasks.

The two experiments test on the artificial intelligence function and the response time of the server, arguably two of the most important components of the application. Due to the fact that Gemini 1.5 artificial intelligence model, a model known for its strict content filtering, is utilized in our program, it is crucial to test the boundaries of such artificial intelligence model. Our first experiment is designed to test that, feeding what some may consider a controversial/inappropriate topic to the Gemini 1.5. The result of the experiment concluded that although Gemini will filter certain topics, it is able to process the majority ofmost other topics. The second experiment tests the response time of the server. Response has been proven to be one of the most important factors contributing to user satisfaction, thus testing the response time of the server is crucial for a success for the program. Our server response time has proven to be consistent and reliable.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. The Graphic Design

The Chrome Extension front end that was designed for the Focus project is intended to act as a convenient and easy to use user interface. Additionally, to include the proper function with the

back end server and general requirements for the extension, the use of background javascript for handling and displaying resources was employed. One of the most significant challenges in the front end is the graphic design; the initial design is simplistic and ordinary, it took a couple trials with different designs to discover the most attractive choice. Coming up with the suitable color, font, and the placement of different sections is crucial for the user experience. The background javascript itself was also challenging, due to the nature of the extension having quite a numerous amount of moving parts.

2.2. The Selection of the Chat Model

To include the usage of the Gemini learning language model, a backend server was employed to maintain the simultaneous analysis of the user's browsing history and patterns. Within the backend server, multiple different Gemini prompt functions exist to perform the tasks of analyzing the user's browsing habits and provide support and advice. During the development phase, the biggest challenge we encountered was the selection of the chat model, and the experimentation that took place. During this experimentation, we performed analysis of multiple different chat models, including ChatGPT [refer to chat GPT], which we found to restrict the API access due to too many simultaneous requests. Gemini proved to be a good alternative, as it proved to be much more accessible without sacrificing accuracy.

2.3. Errors

The Flask server is the backbone of this project. It links the backend algorithm, which accesses artificial intelligence, to the frontend display. It also creates a set structure and defines the ins and outs of resources for the project. Due to unfamiliarity with the functionalities of a Flask server, the most significant challenge while implementing the backend server was encountering bugs such as syntax errors during the definition of the various inputs and output routes. Additionally during development, it was important that we keep in mind the intended hosting provider for the project, Render.com, as they have their own set rules and configuration guides.

3. SOLUTION

Taking a look at the overall functionality of the Focus Chrome extension, it comprises three major components. These three major components include the extension front end available on the Chrome Web Store, the back-end Flask server, and the AI model back-end functions. Once installed, the user has the option available on the extension popup to start the collection of their browsing habits. This data will be sent to and be processed by the back end server, which will then be fed to the AI model back-end function to analyze the user's data. The AI model back-end function will output a percentage between 0 and 100, indicating how productive the user is, with 0 percent being the most productive and 100 percent being definitely time wasting. This data is then returned back to the user as a response from the server, and stored in the user's Chrome local storage. After the user has collected a good amount of data regarding how they are currently using the browser, they then have the option to send they're collected data to another back-end behavioral analysis AI model function [4]. When the user clicks on the 'Analysis' button, the user is directed to an options page where they can see they're screen time for different tabs they have open, as well as a button to analyze their browsing behavior. When the user clicks the button, their browsing data is sent to the AI model analysis function on the server, where it is processed and returned back to the user, giving them a complete analysis on their productivity, ways they got distracted, and advice on how to be more productive.

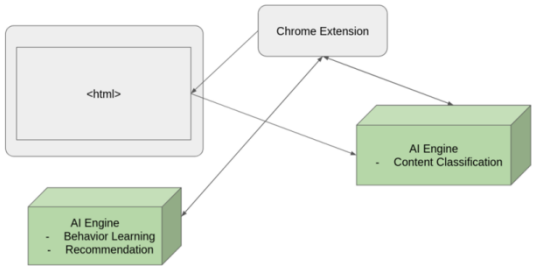

Figure 1. Overview of the solution

The front-end extension directly communicates with the user, giving users the options to turn on screen time analysis, kick-starting the background javascript code to communicate with the back-end server on a set interval of 10 seconds. User information such as the tab title, URL, and text samples will be sent to the back-end. The result from the back-end will be stored in chrome storage, which will be displayed by the front-end.

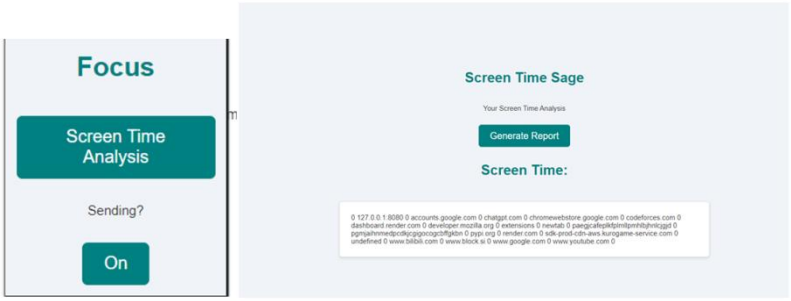

Figure 2. Screenshot of analysis

```
setInterval(function() {
  if (sendData) {
    chrome.tabs.query({ active: true, currentWindow: true }, function(tabs) {
      if (tabs && tabs[0]) {
        const tabId = tabs[0].id;
        const title = tabs[0].title;
        const url = tabs[0].url;

        chrome.scripting.executeScript(
          {
            target: { tabId: tabId },
            func: () => {
              const elements = document.querySelectorAll('h1, h2, h3, p, li, a');
              const elementContents = Array.from(elements).map(element => {
                if (element.tagName.toLowerCase() === 'a') {
                  return element.href;
                } else {
                  return element.textContent;
                }
              });
              chrome.storage.local.set({ content: elementContents });
            }
          },
          () => {
            chrome.storage.local.get('content', function(data) {
              const postData = { title: title, url: url, text: data.content };

              sendHTTPPostRequest('https://focus-1-3izr.onrender.com/askContent', postData, function(error, response) {
                if (error) {
                  console.log(error);
                  alert(error);
                } else {
                  // Update popup.js
                  chrome.runtime.sendMessage({ action: "ResponseReceived", message: response });

                  // Store the response data
                  chrome.storage.local.get('stored_data', function(currentData) {
                    const storedData_json = currentData['stored_data'] || '';
                    const newData = storedData_json + JSON.stringify(response);
                    chrome.storage.local.set({ 'stored_data': newData });
                  });
                }
              });
            });
          }
        );
      }
    });
  }
}, 10 * 1000);
```

Figure 3. Screenshot of code 1

In the code depicted above, a setInterval is created to send data to the back-end server if a set condition, sendData, is met. If sendData, a boolean value set to be true or false depending on whether the user has the option to send the data on or off, is evaluated to true then a query is made to Chrome's tabs [5]. In this query, the tabId, title of the webpage, url, and parsed html are gathered from the current active tab. With all of that data, a request to the back-end server is made, and a response of whether they are being productive or not is returned. All of this data is then taken and a new entry is stored in Chrome's local storage, where all of these items are stored as a set making a log of the user's current activity. Over time, these are collected and saved in Chrome's local storage.

The next major component, the back-end flask server, is responsible for the connection between the front end chrome extension and the AI functions capable of providing analysis on the user's web behavior. This server utilizes the Flask library for facilitating http requests from the user's browser to the back-end server.

Figure 4. Screenshot of code 2

As the backbone of the server, we chose the Flask framework as it supports establishing an easy to use HTTP server while working natively in Python [6]. Flask is a micro web framework that can establish a HTTP server that can be hosted on the open internet. It is convenient that this server be hosted on the internet as it can act as a central location where all devices using the extension can send their requests and use the same model. This promotes the conservation of resources and also allows us to monitor the health of the AI model functions and users. To facilitate the routing of user requests to each function, we created and utilized multiple server routes using the flask library, namely the '/askContent' and '/askBehaviour' routes as seen in the code screenshot above. In both of the major functions, the flow of data is quite simple. The data

is received at the request and 'grabbed using request.get_data(),' upon which it is immediately to the respective background function 'askGemWithHistory' or 'process_data_and_ask_chatGPT.' While the server is responsible for receiving the requests and passing it to the back-end functions, the back-end functions themselves are responsible for the processing of the user data and drawing of conclusions. Each back-end works a little bit differently, however, they all function by utilizing prompt engineering with a customized prompt.

```
def askGem(url,text,title):
  genai.configure(api_key="***APIKEY***")

  model = genai.GenerativeModel('gemini-1.0-pro-latest')
  chat = model.start_chat()

  response = chat.send_message("""Hey Gemini, here is some information of a webpage: the url: {}.
    This is the text description of the content: {}
    This is the title of the content: {}.
    Determine if the content is time-wasting or productive.
    Please respond with an integer of percentage score of the page indicating its time-wastingness. 0 percent being definitely productive, 100
    A sample response is: 10
    Do not give anything other than an integer such as the example above""".format(url,text,title))
  response.resolve()

  return int(response.text)
```

Figure 5. Screenshot of code 3

Observing the back-end function above, the askGem function, the major steps that we're repeated for all of back-end functions can be seen. The very first thing that must be taken care of in order to utilize the Gemini API is the establishment of the API key. ChatGPT also requires an API key, however, throughout this project we learned to favor Gemini as it is entirely free and has a shorter cooldown for back to back requests [7]. After the API key is set up, the design of the prompt can be put into play. When designing the prompt, it was important to us to keep the instructions as simple and as clear as possible, as we encountered a lot of issues with the LLM misinterpreting or oftentimes completely ignoring instructions. We found that by keeping the instructions clear cut, the LLM worked best. For this prompt, as the intention is to classify a web page that the user is on as productive or unproductive, the prompt is designed to allow inputs for certain web page elements such as the url, title, and text description and asks the model to give a rating of how productive the web page seems.

4. EXPERIMENT

4.1. Experiment 1

One potential point of concern that we identified during the development process was how the Gemini model will return uninformative responses if what it deems is inappropriate content is identified in the prompt. This could be a major issue, as if Gemini encounters inappropriate content, it could return data that is uninformative and is no help to the behavioral analysis prompt. To test to see how relevant of an issue this is, we will perform an experiment that is designed to see how and what webpages, that we have deemed are appropriate and productive, will be flagged by Gemini as inappropriate. The experiment will encompass using the extension on five different web pages we believe may trigger this irregular response from Gemini, and we will log if Gemini responds irregularly in our data. This will allow us to detect what specific content Gemini doesn't like and if this will be a problem for users of the extension.

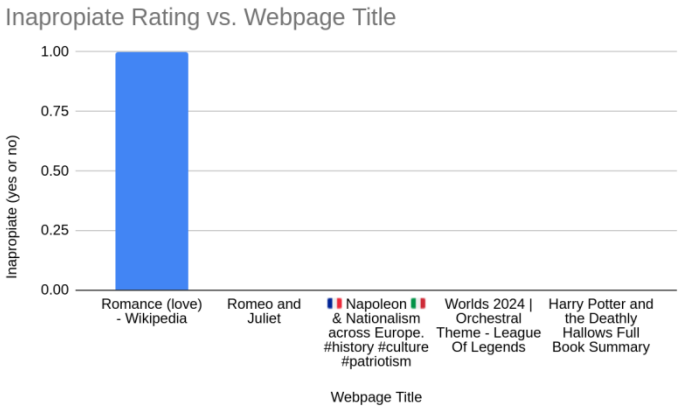

Figure 6. Figure of experiment 1

After performing the experiment, we found that there was only really one instance in which we encountered the Gemini model to return a list of safety ratings, indicating that the request that was made was deemed inappropriate. The reason that the Gemini model returned this rating was due to the "sexually explicit language" that was found in the page, however, the page included any instances of these in a professional and educational manner. The fact of the matter is, it is possible for a student to be navigating here with the intention of being productive, however, Gemini will encounter this issue and return a useless response. We figure the reason this response is returned is because Gemini is designed to be quite strict when it comes to any kind of content it deems is inappropriate, therefore, it naturally doesn't want to touch it. However, what surprised us was how the Gemini model seemed to not deem violence to be inappropriate, despite how much was in the Harry Potter book summary in the experiment.

4.2. Experiment 2

For any application, particularly one that prides itself on consistency and reliability, a back-end server that is durable and returns consistent results is of great importance. As such, we are interested in testing to ensure that the back-end server that we have created is indeed strong and consistent in return times.

To test the consistency of the back-end server we have designed an experiment with the intention of performing repeated requests to see both how quickly the back-end server formulates a response and returns it as well as if there are any reliability issues with the hosting provider we are using. The design of this experiment is quite simple, it involves two team members - one for performing the requests and another for keeping track of the time the request takes. The first team member, upon sending the request, informs the second team member that they have sent it, upon which the second team member will start the timer. Upon receiving the request, the first team member will again let the second team member know and they will subsequently stop the timer. Finally, the durations will be recorded and laid out onto a chart for easy analysis.

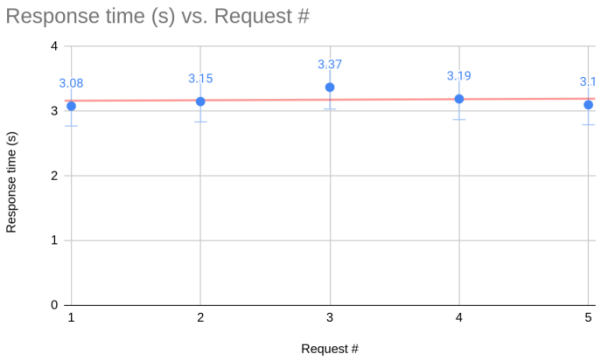

Figure 7. Figure of experiment 2

After performing the experiment, we are quite impressed with not only the reliability of the back-end server that we are using, but also how consistent the response times were as all were around 3 seconds. Analyzing the results, the average response time was 3.178 seconds which we are quite happy with as it is quite a quick response time. The most surprising bit to us as the research team is how consistent the request times were. We believe this response time is due mostly to how simple and efficient the back-end server is formatted to be, as well as how the prompt elicits a clean response from the LLM [8]. Both of these factors were designed to keep the time at each step small, in order to promote a quicker back-end while not compromising on quality of generation. Additionally, the response time could be do to the lack of input data for the back-end model to receive, however, we are quite confident in the server and back-end server prompts ability to quickly handle these heavier requests after reviewing the results of this experiment.

5. RELATED WORK

One major methodology of keeping users informed of how productive they are being on their device is Apple Screen Time [9]. Apple Screen Time is a household name service used in both families and work environments for keeping track of what apps or services a user uses on their mobile device. While Apple Screen Time remains effective in its basic function of tracking time spent on certain apps, its features are quite lackluster and uninformative of whether the user was being productive or not. The Focus Chrome extension improves upon the Apple Screen Time service by providing in depth AI analysis of what the user is currently working on, and even gives tips on how to personally improve one's productivity.

Another method of helping keep users engaged and focused on their tasks is the LifeAt desktop application [10]. The LifeAt desktop application is a desktop manager system that seeks to provide a digital environment designed to keep the users engaged and on task. LifeAt does this by keeping only work tools on the desktop, and providing an easy to use interface for users to navigate their work. On the surface, LifeAt provides a great desktop environment to keep users engaged, however, it lacks reflective feedback based on how the user keeps engaged and what they can work on personally. The Focus Chrome extension proposed in this paper aims to solve this issue by providing customized feedback based on the user's personal interaction with the browser.

Blocksi is another application designed for any operating system to keep students engaged by enabling teachers with tools to monitor their device activity [11]. Additionally, Blocksi also has newer custom adaptive AI features designed to filter content and keep users from getting off task. While Blocksi is effective in keeping users productive in the short term, they lack in teaching

users how to stay productive themselves. Focus aims to fix this problem by providing users custom feedback to help keep themselves responsible.

6. CONCLUSIONS

While the Screen Time Sage extension works well currently, there are a few areas the project could improve, namely, in the front end user experience as well as the quality of the AI generations [14]. The user interface for the extension is a little bare bones in some parts, as the backend functionalities were definitely the priority during the development process. In the future, we seek to employ more front end development strategies and completely redesign the front end to be quite more user friendly. Additionally, while the AI model generations are definitely informative for the user, they do sometimes tend to get off track with the content and purpose of the prompt. To solve this issue, using an improved model from ChatGPT, such as the 4.0 model or even using a custom model trained to give tailored feedback could be strategies we employ for creating better and more consistent generations [15].

Overall, the Screen Time Sage extension provides a simple and straightforward approach to analyze one's screen usage utilizing AI to provide an objective feedback to the user. It is well simple and well-structured, although the AI functions may be improved.

REFERENCES

[1] Braghieri, Luca, Ro'ee Levy, and Alexey Makarin. "Social media and mental health." American Economic Review 112.11 (2022): 3660-3693.

[2] Tiller, Hanna. "Analysis of the function time limits within the IPhone feature screen time." Conference in interaction technology and design.

[3] Tiller, Hanna. "Analysis of the function time limits within the IPhone feature screen time." Conference in interaction technology and design.

[4] Jewitt, Carey, et al. "School use of learning platforms and associated technologies." Institute of Education–University of London (2010).

[5] Team, Gemini, et al. "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context." arXiv preprint arXiv:2403.05530 (2024).

[6] Mufid, Mohammad Robihul, et al. "Design an mvc model using python for flask framework development." 2019 International Electronics Symposium (IES). IEEE, 2019.

[7] Brito, Allan. Blender Quick Start Guide: 3D modeling, animation, and render with Eevee in Blender 2.8. Packt Publishing Ltd, 2018.

[8] Carlini, Nicholas, Adrienne Porter Felt, and David Wagner. "An evaluation of the google chrome extension security architecture." 21st USENIX Security Symposium (USENIX Security 12). 2012.

[9] Giray, Louie. "Prompt engineering with ChatGPT: a guide for academic writers." Annals of biomedical engineering 51.12 (2023): 2629-2633.

[10] Marotta, Veronica, and Alessandro Acquisti. "Online distractions, website blockers, and economic productivity: A randomized field experiment." Preliminary Draft (2017).

[11] Lottridge, Danielle, et al. "Browser design impacts multitasking." Proceedings of the human factors and Ergonomics Society Annual Meeting. Vol. 56. No. 1. Sage CA: Los Angeles, CA: SAGE Publications, 2012.

[12] Carlini, Nicholas, Adrienne Porter Felt, and David Wagner. "An evaluation of the google chrome extension security architecture." 21st USENIX Security Symposium (USENIX Security 12). 2012.

[13] Islam, Raisa, and Imtiaz Ahmed. "Gemini-the most powerful LLM: Myth or Truth." 2024 5th Information Communication Technologies Conference (ICTC). IEEE, 2024.

[14] Zhang, Bo, Jun Zhu, and Hang Su. "Toward the third generation artificial intelligence." Science China Information Sciences 66.2 (2023): 121101.

[15] An, J., W. Ding, and C. Lin. "ChatGPT." tackle the growing carbon footprint of generative AI 615 (2023): 586.