Sustainable Software Engineering Project 2: SusMode

Arnav Biswas
MSc. Computer Science
TU Delft
Student number: 5721962
Email: arnavbiswas@tudelft.nl

Benas Pranauskas
MSc. Computer Science
TU Delft
Student number: 6445667
Email: b.pranauskas@student.tudelft.nl

Ignas Vasiliauskas
MSc. Data Science & AI Technology
TU Delft
Student number: 5291682
Email: i.vasiliauskas@student.tudelft.nl

Jeffrey Meerovici Goryn
MSc. Computer Science
TU Delft
Student number: 5730783
Email: j.g.meerovicigoryn@student.tudelft.nl

Nicolas Hornea
MSc. Data Science & AI Technology
TU Delft
Student number: 5093597
Email: A.N.Hornea-1@student.tudelft.nl

***Abstract*—This paper presents SusMode, a browser extension designed to promote sustainable web browsing by reducing the unnecessary energy consumption caused by modern web content. Contemporary websites often include resource-heavy features, such as advertisements, tracking scripts, autoplay media, and AI-driven features, many of which provide limited user value while increasing power consumption. SusMode addresses this issue through a set of user-configurable buttons that enable and disable these computationally-intensive features.**

Due to the fact that energy measurement is not possible through a browser environment, the extension uses an estimation-based approach. SusMode also provides user feedback through counters that encourage energy-aware browsing behaviour.

An experimental evaluation comparing browsing sessions with and without SusMode shows that the extension achieves a statistically significant reduction in CPU energy consumption (4.34%) and energy-delay product (4.59%) without negatively affecting performance. Although reductions in network and total energy consumption are observed, they are not statistically significant due to the high variance across websites.

Overall, the results demonstrate that lightweight, client-side interventions can meaningfully reduce the energy footprint of everyday web browsing. SusMode highlights the potential of browser extensions as practical tools for sustainable software engineering, enabling users to make more environmentally conscious decisions with minimal impact on usability.

I. Introduction

Web browsing is often perceived as a lightweight digital activity, yet modern websites can trigger substantial resource consumption on both client devices and remote infrastructure. Pages frequently load a large number of scripts, advertisements, trackers, autoplay media, and interactive components. Many of these provide limited value to users while still consuming computational power, bandwidth, and energy [6]. In addition, recent browser-integrated AI features and search-engine AI assistants introduce additional processing and back-end demand, further increasing the environmental footprint of daily browsing.

This project investigates how sustainable software engineering principles can be applied to the browsing experience through a browser extension called *SusMode*. The goal of SusMode is to help users reduce unnecessary digital energy consumption by enabling a *sustainable mode* in which selected high-impact features are limited or disabled. In its current design, the extension can suppress AI-related search elements, block advertisements, reduce script-heavy content, and pause or hide video content. By reducing unnecessary page activity, the extension aims to encourage more energy-aware browsing behaviour without requiring substantial effort from the user.

A central challenge in this work is that direct measurement of browser energy consumption is not possible from within a standard browser extension. Browser extensions do not have access to hardware-level telemetry, such as CPU package power, GPU power, or per-process battery drain. Therefore, rather than measuring energy use directly, SusMode adopts an estimation-based approach. The extension uses observable browser-side indicators, such as loaded resources, script activity, media playback, and blocked high-cost features, as proxies for energy demand. These proxy services provide a practical and explainable basis for estimating the relative impact of browsing activity.

The contribution of this project is twofold. First, we design and implement a Chrome extension prototype that applies sustainability-oriented interventions during browsing. Second, we explore how browser-accessible metrics can be used to estimate the energy implications of user actions and page features. Together, these contributions demonstrate how sustainability concerns can be made visible at the level of everyday software interaction.

II. Related Work

Previous research has demonstrated that web browsing can significantly impact energy consumption and that browser-

level optimizations can reduce this impact. This section discusses three relevant works that motivate and inform the design of SusMode.

GreenBrowsing proposes an approach to improving the energy efficiency of web browsing through a Chrome extension and a backend certification system [1]. The system monitors browser resource usage and limits unnecessary background activity, particularly for inactive tabs. The authors demonstrate that reducing unnecessary computation and resource loading can significantly reduce CPU and memory usage while maintaining an acceptable user experience. This work shows that browser extensions can serve as practical sustainability tools. SusMode builds on this idea by focusing on user-configurable interventions such as disabling AI search features, blocking advertisements, limiting scripts, and pausing videos.

The System for Analyzing Mobile Browser Energy Consumption introduces a framework for measuring the energy consumption of individual web components such as JavaScript, CSS, images, and plugins [4]. The system provides a fine-grained analysis of how different page elements contribute to total energy usage. The authors demonstrate that modifying resource-heavy components can significantly reduce energy consumption without negatively affecting the user experience. This work motivates our approach of targeting high-energy components such as scripts, videos, and AI-related features.

Bui et al. rethink the traditional focus on performance optimization in mobile browsers and instead consider energy-performance trade-offs [5]. Their work identifies several sources of energy inefficiency in mobile web page loading, including aggressive resource processing and inefficient rendering strategies. The authors propose adaptive scheduling and resource loading techniques that achieve significant energy savings without increasing page load times. This work supports the idea that reducing unnecessary browser activity can lead to meaningful energy savings, which aligns with the goals of SusMode.

Together, these works demonstrate that browser-level interventions and resource-aware optimization can significantly reduce energy consumption. SusMode extends this research by implementing a lightweight browser extension that both reduces energy-intensive features and estimates browsing-related energy consumption using browser-accessible metrics.

III. Methodology

This project follows a prototype-driven methodology in which SusMode is designed, implemented, and evaluated as a browser extension to reduce unnecessary energy-intensive browsing behaviour. The methodology combines two complementary parts: *intervention* and *measurement*. The intervention part focuses on reducing resource-intensive browser activity through extension-based controls. The measurement part focuses on estimating energy-relevant effects using browser-level counters and external system-level monitoring scripts.

A. System Design

SusMode was implemented as a Google Chrome extension based on Manifest V3. The extension consists of three main components: a pop-up interface, a background service worker, and a content script. The pop-up provides a user-facing control panel where sustainable browsing features can be enabled or disabled. The background service worker manages persistent logic that is not tied to a single page, such as tab suspension and network request filtering. The content script operates inside web pages and performs page-level interventions such as removing AI-related interface elements, blocking advertisements, hiding videos, and suppressing selected scripts.

The extension was designed around the idea of a *sustainable mode*, where users can selectively activate features that reduce unnecessary browser work. In the current prototype, these features include disabled AI-related search elements, blocked advertisements, blocking tracking and scripts related to ads, pausing or hiding videos, suspending inactive tabs, throttling background tabs, and allowing site-specific exceptions through a whitelist.

B. Browser-Side Interventions

The first methodological step was to identify browser behaviours that are likely to contribute to unnecessary energy use. Based on the prior literature and the practical inspection of modern web pages, the following categories were targeted.

AI search suppression. The extension scans the page for interface elements containing phrases such as *AI Mode*, *Search with AI*, and *Ask AI*. When enabled, matching clickable elements are removed from the page. The goal is not to prevent all server-side AI processing, but to reduce user interaction with AI-based browsing features and discourage repeated use of energy-intensive search workflows.

Ad blocking. The content script searches for ad-related page elements using a set of selectors covering common ad slots, sponsored containers, advertising classes, and ad `iframes`. Matching elements are removed from the document. In addition, the background service worker uses Chrome's declarative network request API to block requests to known advertising and tracking domains before they are fully loaded. This combines DOM-level blocking with request-level prevention.

Script blocking. The extension targets selected ad, analytics, and tracking scripts by matching script source URLs against known patterns such as advertising networks, analytics providers, and tracking services. The purpose is to reduce the amount of non-essential client-side computation and background communication associated with browsing.

Video suppression. The extension detects native video elements and embedded video `iframes`. Native videos are paused, stripped of autoplay behaviour, and hidden. Embedded videos are hidden while their requests are additionally constrained through network rules where possible. This feature is intended to reduce energy consumption associated with autoplay media and video-heavy pages.

Background tab optimization. The background script records the last access time of tabs. If a tab remains inactive for more than a predefined timeout and is not pinned, audible, active, or already discarded, it is suspended using Chrome's tab discard mechanism. On the page level, when a tab becomes

hidden, the content script pauses media playback and pauses CSS animations. Together, these actions aim to reduce energy wasted by inactive tabs running in the background.

Whitelist support. Since aggressive blocking may interfere with the normal functionality of some websites, the extension includes a whitelist mechanism. Users can exclude selected domains from SusMode's interventions, allowing the system to remain usable in practice.

C. Tracking Intervention Effects

To make the effects of the extension visible to the user, SusMode maintains cumulative counters for blocked or disabled elements. These include the number of AI-related elements removed, advertisements blocked, scripts blocked, and videos paused or hidden. These counters are stored using Chrome local storage and displayed in the pop-up interface.

The extension also computes a simple aggregate estimate of lifetime energy savings from these counters. In the current prototype, blocked ads, blocked scripts, and blocked videos are assigned heuristic weights, and the cumulative total is displayed as an estimated energy-saving indicator. This estimate is intended as a relative and user-facing sustainability signal rather than a precise physical measurement.

D. External Energy Measurement

Because a browser extension cannot directly access hardware-level power telemetry, the methodology also includes external system-level scripts to observe energy-related behaviour outside the browser sandbox. Three supporting Python tools were developed for this purpose.

CPU energy monitoring. The measure_cpu.py script measures CPU package energy using the Intel RAPL counters method used by González-Cabañas et al. [3], exposed through the Linux powercap interface. The script reads cumulative energy values in microjoules, corrects for counter wraparound, and periodically reports interval energy, cumulative energy, and average power. This allows browser activity to be observed against direct CPU energy measurements on supported Linux systems.

Network energy estimation. The measure_network.py script uses `psutil` to read cumulative sent and received bytes across network interfaces. Transferred data volume is converted into estimated energy using a fixed energy-intensity coefficient expressed in kWh per GB, an approach partially inspired by Baliga et al. [2]. The script reports interval traffic, estimated power, and cumulative energy. This provides a coarse estimate of the energy associated with data transfer during browsing sessions.

Zen mode process cleanup. The zen_mode.py script terminates selected non-essential desktop applications such as media players, chat clients, and other unrelated GUI applications. This was introduced to reduce interference from background processes during measurements and to create a cleaner environment when testing the browser extension.

E. Measurement Procedure

The evaluation is done using two types of session: a control version that runs without the SusMode extension; a treatment version where SusMode is applied. Each is run 50 times in a randomized order that prevents possible bias. The sessions start from a cold-hardware state, and all unnecessary processes are removed by a "Zen mode" script. Each run visits five sites chosen for their popularity in European web users: BBC News, CNN, Reddit, YouTube, and Google Search. The CPU energy and the network energy , both measured in Joules, are computed from the external monitoring scripts. The *Energy Delay Product* (EDP $= E \times t$) combines energy and the duration of the session, and this method is used to penalize sessions that conserve energy at the cost of performance. EDP is also correlated with the values taken by the counters that monitor blocked processes such as advertisements and browser scripts that are blocked by the extension.

F. Methodological Scope and Limitations

The methodology is designed to support a practical sustainability prototype rather than exact hardware-level attribution of browser energy use. The Browser-side counters indicate what kinds of content were blocked or suppressed, but do not directly measure the exact joules saved by each intervention. Likewise, the external CPU and network scripts observe system-level behavior and cannot perfectly isolate only the browser tab under study. Therefore, the methodology should be understood as an estimation-based approach that combines direct intervention logging with coarse-grained system monitoring.

Despite these limitations, this methodology is suitable for the goals of the project. It enables the implementation of a working browser extension, supports transparent user-facing sustainability controls, and provides a reasonable basis for investigating how browser behavior can be shifted toward lower-energy usage.

IV. Implementation

We implemented SusMode as a Google Chrome Manifest V3 extension[1] with three components: a background service worker for network and tab-level controls, a content script for in-page interventions, and a popup user interface (UI) for configuration and feedback. The implementation aims to reduce unnecessary energy-intensive browser activity by preventing costly resources from being requested or executed, suppressing media and AI-related interface elements that tend to induce heavier workloads, and minimizing background-tab work. All feature flags and counters are persisted in `chrome.storage.local` and applied via storage listeners.

[1]As modern Microsoft Edge versions also are Chromium-based, features might be co-operable, but we have not tested the application there and there are still some parts which might rely on the Chrome-only API.

A. Background Controls: Network and Tabs

At start-up and when the main toggle changes, the background worker installs/removes declarative network request (DNR) rules and schedules a periodic alarm for tab suspension. DNR rules are the most important first barier, as these (contrasting DOM actions) prevent these resources from loading altogether.

- **Streamlined search and request blocking.** DNR rules (prioritised to stop downloads early) add `udm=14` to `https://www.google.com/search`, this enables *Web Mode*, which minimizes modern Chrome features like AI summaries and snippets in the results. They also block common advertising/tracking endpoints (e.g., DoubleClick, Google Ads, Criteo, Amazon Ads, Tag Manager/Analytics, Facebook pixel), and block media from ad networks (e.g., `mp4/webm` served via ad CDNs).
- **Tab suspension.** The worker tracks last-access times per tab and, every 5 minutes, discards tabs inactive for 15 minutes (excluding active, pinned, audible, or already discarded tabs). Discarding uses Chrome's native mechanism to reduce CPU and memory for background tabs without data loss.

B. In-Page Interventions

The content script complements DNR with DOM-level actions. A `MutationObserver` re-applies interventions to dynamically inserted content, both scanning and observing are skipped on whitelisted domains.

- **AI removal.** Removes clickable UI containing elements such as "AI Mode", "Search with AI", and "Ask AI".
- **Ad removal.** Deletes common ad slots, sponsored containers, and ad `iframes` (e.g., DoubleClick, Google Syndication), reducing on-page CPU and layout work in addition to blocked requests.
- **Script removal.** Drops advertising/analytics scripts from typical sources (e.g., DoubleClick, Google Ads/Analytics/Tag Manager, Criteo, Amazon Ads, Facebook, Pinterest, Segment, Mixpanel, Hotjar, etc.) to minimise non-essential computation and background traffic.
- **Video removal.** Pauses and hides native video and hides embedded players (YouTube/Vimeo/Dailymotion).

C. Background-Tab Optimisation

When a tab becomes hidden and throttling is enabled, the script pauses audio/video and sets CSS `animationPlayState` to paused; these are restored on visibility. To preserve site compatibility, no aggressive timer is applied to tabs.

D. User Interface and Energy Counters

All toggles (global and per-feature), suspension/throttling flags, counters, and a domain whitelist are stored locally. On initialisation, the content script loads state, scans and updates counters. The popup exposes:

- A global on/off switch (installs/removes DNR rules; reloads the active tab when turning off).
- Per-feature toggles (AI, ads, scripts, videos), tab suspension, and throttling.
- Whitelist management for the current site and a modal to review/remove entries.
- Live metrics and a heuristic energy indicator. The indicator applies fixed weights (ads: 0.5 kJ, scripts: 1 kJ, videos: 5 kJ) to display an indicative cumulative kWh figure.

V. Results

Before a more in depth statistical analysis, each metric was assessed for normality using the Shapiro-Wilk test. As shown in Fig. 1, all metrics exhibit non-normal distributions ($p < 0.001$), which is represented by right-skewed tails due to occasional high-energy spikes. The Mann-Whitney U test was used for all pairwise comparisons. Out 100 sessions, 10 runs were flagged as outliers after computing the z-score (threshold $= 3$) and resampled. Results are considered statistically significant at $\alpha = 0.05$.

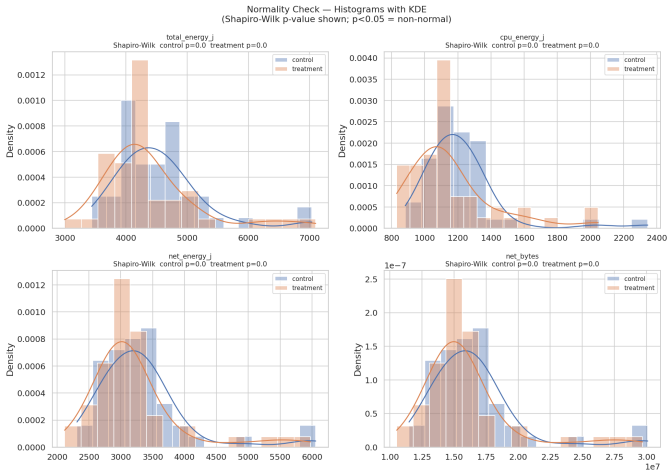

Fig. 1. Normality check — histograms with KDE for all metrics. Shapiro-Wilk $p < 0.001$ for all distributions.

A. Effect of SusMode on Energy Metrics

Fig. 2 shows the full energy distributions for control and treatment across all three energy metrics. The treatment violin for CPU energy sits visibly lower than control, while total and network energy distributions overlap considerably.

The extension produces a relevant reduction in CPU energy of 4.34% ($-52.8\,\mathrm{J}$ per session, $p = 0.030$). Total energy and network energy decrease by around 3%, but neither reaches a significant value. The network standard deviation ($\sim 730\,\mathrm{J}$) is substantially larger than the observed difference ($\sim 102\,\mathrm{J}$), meaning the signal is present but it is difficult to estimate at 50 runs. Fig. 3 represents the effect sizes across all metrics; every metric is negative, indicating that the susmode sessions are never below the control.

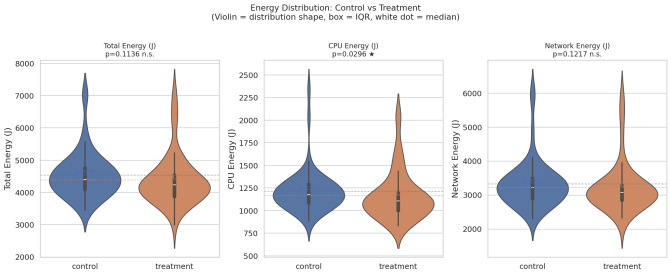

Fig. 2. Energy distributions for control and treatment conditions. White dot = median, thick bar = IQR.

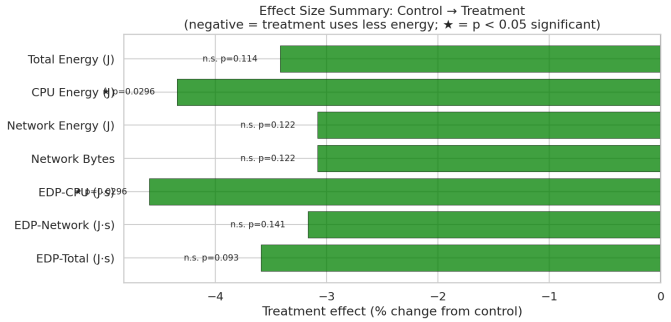

Fig. 3. Effect size summary across all metrics. Negative values indicate the treatment uses less energy. Starred bars are statistically significant ($p < 0.05$).

B. Energy Delay Product

Fig. 4 shows the EDP box plots for all three conditions. EDP-CPU drops by 4.59% ($p = 0.030$), similar to the energy. Considering that the session duration is nearly identical across conditions ($\sim$71–72 s), we can confirm that SusMode reduces CPU load without meaningfully affecting the browser's performance.

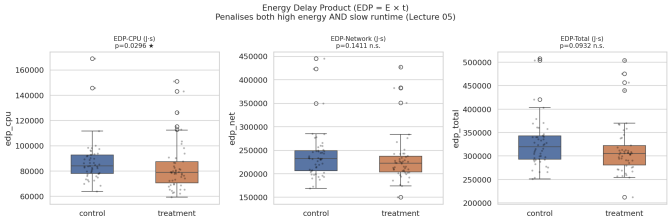

Fig. 4. Energy Delay Product (EDP $= E \times t$) for CPU, network, and total energy.

C. Per-Site Analysis

Fig. 5 breaks down network byte reductions by site. CNN shows the largest reduction at -11.3% ($p = 0.052$, borderline), because of its large use of third-party ads and tracker scripts which are blocked by SusMode. In turn, BBC News shows less savings as it runs much less blockable background processes. YouTube and Reddit exhibit high variance due to their dynamic content. This masks the modest savings the extension achieves.

‘‘‘

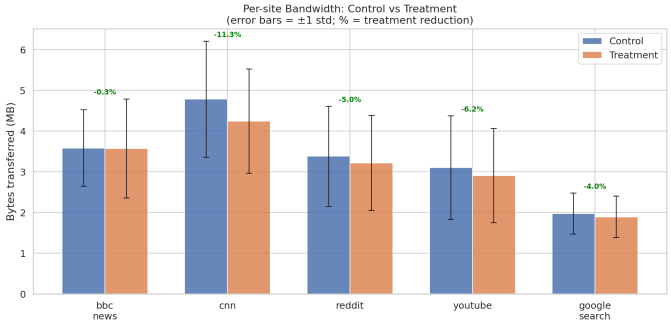

Fig. 5. Per-site bandwidth reduction (control vs. treatment). Error bars show ± 1 standard deviation.

VI. Discussion

This study provides evidence that browser intervention can lead to a significant reduction in energy consumption, particularly at the CPU level. The observed decrease of 4.34% in CPU energy, along with a comparable reduction in the CPU Energy Delay Product (EDP), suggests that SusMode effectively reduces energy consumption without negatively impacting user-perceived performance. This aligns with previous expectations that restricting scripts, advertisements, and autoplay media can reduce overhead while maintaining responsiveness.

One of the main takeaways is that while CPU energy consumption decreases, the reductions in network and overall energy use are smaller and not statistically significant. Although all metrics trend in a positive direction, network energy varies a lot—especially on dynamic platforms like YouTube and Reddit—which makes it harder to detect clear effects with the current sample size. This suggests that savings in network energy depend more on context and the type of content being loaded, while CPU savings are more consistent. In particular, sites that rely heavily on third-party advertisements and tracking, such as CNN, seem to offer greater room for improvement, supporting the idea that the effectiveness of interventions depends on the specific website.

The per-site analysis further highlights that the impact of SusMode strongly depends on the structure and content of individual websites. Pages with a high density of advertisements, trackers, and embedded media benefit the most from the extension's blocking mechanisms. In contrast, more lightweight or static websites exhibit smaller gains, as there is less unnecessary activity to eliminate. This shows that a one-size-fits-all approach to sustainable browsing is inherently limited and that adaptive or site-aware optimisation strategies could further enhance effectiveness.

Another key part of this work is the estimation of energy use in the browser using proxy measurements. Although direct measurement of energy consumption is not feasible inside a browser extension, the use of observable indicators—such as blocked scripts, removed ads, and paused videos—provides a transparent and user-understandable approximation of impact. Although these estimates are not precise, they serve an important role in raising user awareness and encouraging

more sustainable interaction patterns. The combination of these proxy metrics with external system-level measurements strengthens the overall evaluation by bridging the gap between user-facing feedback and actual hardware-level effects.

The lack of performance drop—shown by almost identical session times in both the control and treatment groups—is especially important. It suggests that sustainability does not have to come at the cost of usability. This is important for real-world adoption, as users are unlikely to accept solutions that noticeably degrade their browsing experience. SusMode demonstrates that meaningful reductions in energy use can be achieved in a largely unobtrusive manner.

At a broader level, the findings support the idea that sustainability can be integrated into everyday software systems through relatively lightweight mechanisms. Rather than requiring fundamental changes to web infrastructure, tools like SusMode empower end users to reduce their digital energy footprint through configurable client-side controls. This positions browser extensions as a practical and scalable way to promote sustainable software practices.

However, the results also indicate that the achievable savings at the individual session level are limited. Although a reduction of 3 to 5% is small in isolation, the cumulative effect between millions of users and browsing sessions could be substantial. This highlights the importance of considering both micro-level efficiency gains and their macro-level implications.

Overall, these results show that SusMode works as a practical first step toward more sustainable browsing. It manages to reduce energy use—especially on the CPU side—without slowing things down or making the browsing experience worse, which is crucial for real life adoption. The extension also fits well with a user-centred approach, since it gives people simple controls to limit unnecessary content without requiring much effort or technical knowledge.

VII. Limitations

There are several limitations to the project, which we acknowledge and address in this section.

First, the methodology for measuring the results of energy usage is constrained by the design of a modern Chrome browser. Browser extensions operate within a sandbox, therefore, lacking direct access to hardware resource usage, such as battery drainage. Consequently, the live version of SusMode relies on fixed heuristic weights to calculate and display the expected energy impact of the extension to the user. It is rather difficult to accurately measure the energy impact of each individual feature of the extension, but designing an approach to carry out such an experiment could be a topic for future research.

Furthermore, to validate the extension's performance, we used custom measuring scripts run in Zen Mode, as described in section III; however, this approach does not perfectly isolate the energy consumption of a single browser tab. Even with Zen Mode enabled and all non-essential processes terminated, the results still include energy consumed by background OS processes.

In addition, we must make a distinction between the measured impact of client-side interventions and the total impact on the environment achieved by the extension. SusMode aims to optimize the client-side of browsing, but it is not able to fully mitigate the back-end processing that may take place even when UI elements are hidden. In contrast, some features (e.g., disabling AI search mode) are intended to influence the behaviour of the users towards more sustainable digital consumption (in this example, prioritizing traditional indexed search over resource-heavy generative AI). The behavioural aspect is difficult to quantify, but it represents a significant part of the project's goal to make the impact of digital choices on the environment visible and actionable.

VIII. Conclusion

SusMode shows that browser-level interventions can effectively reduce energy consumption by suppressing features such as AI search, advertisements, background scripts, as well as pausing videos, suspending inactive tabs, and enabling background throttling. The experimental results confirm the effect of the extension, showing statistically significant reductions in CPU energy demand (4.34%) and the energy-delay product (4.59%) without compromising browser performance or session speed.

Moving forward, the SusMode extension could be expanded beyond a user-facing tool to include a developer-side framework and ultimately help achieve the goal of making sustainable energy choices while browsing clearly actionable. This would allow websites to integrate sustainability decisions into their design workflows, and give clear indications on the impact of the design practices (e.g., how much energy is estimated to be saved by using a script-light alternative version of a website). We therefore publish SusMode as an open-source project on GitHub[2], and aim to collectively extend and further refine it together with the developer community. As such, we hope the project will have a positive impact on reducing the carbon footprint through transparent, shared software engineering standards.

Acknowledgment

The authors would like to thank the teaching team for the course CS4575 Sustainable Software Engineering at Delft University of Technology for continuous support throughout the project. We would like to extend a special thanks to the teaching assistant Andreea Mocanu for direct supervision, and professors Carolin Brandt, Enrique Barba Roque, and Luís Cruz for the high-level guidance and equipping the team with the necessary material and knowledge to successfully navigate the complexities of digital sustainability.

References

[1] G. Avelar and L. Veiga, "GreenBrowsing: Towards Energy Efficiency in Browsing Experience," in *Distributed Applications and Interoperable Systems*, Lecture Notes in Computer Science, vol. 8460, Springer, 2014.

[2]https://github.com/IgnasVas/susmode

[2] J. Baliga, R. Ayre, K. Hinton, W. V. Sorin, and R. S. Tucker, "Energy Consumption in Optical IP Networks," *J. Lightwave Technol.*, vol. 27, pp. 2391–2403, 2009.

[3] J. González-Cabañas, P. Callejo, R. Cuevas, S. Svartberg, T. Torjesen, Á. Cuevas, A. Pastor, and M. Kotila, "CarbonTag: A Browser-Based Method for Approximating Energy Consumption of Online Ads," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 4, pp. 739–750, Oct. 2023. doi: 10.1109/TSUSC.2023.3286916.

[4] Stanford University, "System for Analyzing Mobile Browser Energy Consumption," Stanford Technology Licensing Office. [Online]. Available: https://techfinder.stanford.edu/technology/system-analyzing-mobile-browser-energy-consumption

[5] D. H. Bui, Y. Liu, H. Kim, I. Shin, and F. Zhao, "Rethinking Energy-Performance Trade-Off in Mobile Web Page Loading," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015.

[6] N. Heitmann, B. Pirker, S. Park, and S. Chakraborty, "Towards Building Better Mobile Web Browsers for Ad Blocking: The Energy Perspective (WiP Paper)," in *Proceedings of the 21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, 2020.