Understand the Impact of Infrequently Used Interactions in Webpages with VMuzeel

Linh Tran
Computer Science, NYUAD
lkt4560@nyu.edu

Advised by: Yasir Zaki

ABSTRACT

The long-term issue of dead codes, which is defined as unused codes in webpages, is addressed by various solutions such as Muzeel and Lacuna. However, while these tools have successfully eliminated the majority of dead codes in web applications, leading to significant improvements in page performance, not all remaining codes are strictly necessary for users. Such extraneous codes, which bloat the webpage similarly to dead codes, could be eradicated to further enhance the page load time. Yet, fault identification of the infrequently used codes can lead to a reduction in page usability. This paper aims to correctly identify the less frequently used codes in terms of user interactivity using VMuzeel as an extension of Muzeel, which was originally developed to only eliminate dead codes. The identification of infrequently used code can be carried out by utilizing a computer vision and large language model to identify correctly the possible interactive elements on a website. This comprehensive approach aims to provide insights into optimizing web page performance utilizing the computer vision and LLM models to maintain user experience by efficiently managing function usage throughout the browsing session.

KEYWORDS

computer vision, large language model, Muzeel, network optimization

Reference Format:
Linh Tran. 2025. Understand the Impact of Infrequently Used Interactions in Webpages with VMuzeel. In *NYUAD Capstone Project 2 Reports, Spring 2025, Abu Dhabi, UAE.* 8 pages.

1 INTRODUCTION

Every webpage has different functionalities that allow users to perform various interactions. Not all of which will be executed by users, either because they are invisible on the webpage or because they require complex interactions from users. Hence, these potentially infrequently used codes are wastefully downloaded from the server, reducing page performance generally. To target this problem, this paper aims to 1) effectively eliminate infrequently used codes by identifying the infrequently used interactions on web applications, and 2) analyze the effectiveness of this elimination on the general performance of webpages, as well as on data savings and Mobile Web usage.

The elimination of infrequently used codes is expected to have positive effects on webpage performance, based on previous Muzeel evaluations of dead code elimination. Further reduction of infrequently used codes will also improve data and performance savings, as webpages will only download necessary resources [3]. However, a clear challenge with this method is that if users interact with eliminated codes, it may risk the website crashing. Therefore, it is important to have a method to correctly identify infrequently used codes to minimize this risk. We will investigate how to achieve this while also maintaining majority of webpage functionalities.

Similar to the challenges of identifying dead codes, the dynamic rendering of JavaScript poses difficulties in categorizing infrequently used codes. Firstly, JavaScript allows DOM elements to be added asynchronously, which means these elements may be interacted with at a later time or the interactive elements may be added to the JavaScript after series of interactions. This scenario may lead to the misidentification of the interactive elements as those DOM elements may not exist at the point of identifying the interaction. Secondly, it is unpredictable how users will interact with webpages [5]. Therefore, we aim to analyze user interactions across websites with varied functionalities to identify a general method that can accurately eliminate infrequently used codes for any web application.

We aim to identify the essential interactive elements of a webpage using machine learning models capable of detecting and classifying DOM elements. We use VMuzeel to assess the impact of infrequently used interactions on webpage performance. Specifically, we investigate whether further performance improvements can be achieved on webpages that have already been optimized by Muzeel, i.e., pages from which dead codes have been removed. Additionally, VMuzeel is designed to preserve the visual integrity of the webpage and minimize the risk of crashes by maximizing the accuracy of predictions related to infrequently used code.

In summary, VMuzeel aims to:

- Introduce a novel approach that leverages computer vision and large language models for intelligent code elimination and web optimization.
- Evaluate the effectiveness of VMuzeel in improving load time and overall performance compared to Muzeel, with a particular focus on infrequently used interactions.

2 RELATED WORK

Several attempts have been made to address the dead code issue in webpages. These tools can be used as references for methodologies of eliminating infrequently used code due to their similarities in challenges. An example of these tools is Lacuna, which uses a program analysis technique to identify unused codes in a web application. The proposed approach in Lacuna applies the call graph of the source to identify a series of functions being called from the global scope [5]. Any function not activated from the global scope is defined as dead code and eliminated. While Lacuna does not specifically target infrequently used codes, it provides a backbone idea of hypothetical methods in this paper. WebMedic is one of the few tools that attempts to eliminate less useful codes as dead codes. The WebMedic paper shows that the elimination of these codes has a positive impact on the memory usage of the webpages. However, the eradication of these codes has resulted in a significant reduction in webpage usability [6]. While this does not align with the objective of this paper, which aims to maintain complete usability of processed webpages, WebMedic provides insight into how we can identify infrequently used codes.

Since we are considering user interactivity to further improve webpage performance, it is important to understand how to correctly evaluate the efficiency of the tool because of the predictable decrease in webpage usability. Both papers [2] and [7] provide metrics to understand the effectiveness of web optimization in terms of user interactivity. More specifically, paper [7] introduces a new metric of "time-to-interactivity", measuring how quickly a webpage becomes interactive for users. This metric provides insights into changes in user interaction post-optimization. Paper [2] identifies the level of user interest in different parts of the webpage using gaze tracking. Such analysis may provide potential metrics to determine the threshold of codes that are less likely to be used on webpages. Hence, these tools can be useful in evaluating the efficiency of infrequently used code elimination. Finally, Florence 2 will be our main computer vision model for user interaction analysis due to its light-weight and robustness [9].

Lastly, Muzeel is the main source of codes on which we will implement the infrequently used code elimination. This development will act as an extension of the tool to further reduce more codes and evaluate its effectiveness compared to Muzeel. More significantly, Muzeel applies a black-box approach to identify most dead codes while also considering user interactions. Its basic idea involves exploring all possible states of the website through user interaction emulation and browser event tracking. Reportedly, Muzeel has been tested on a much larger dataset compared to Lacuna and has achieved greater efficiency in identifying and eliminating dead codes [3]. Therefore, the technical implementation of Muzeel will be the basis for evaluating any further implementation of identifying infrequently used codes.

3 METHODOLOGY

3.1 User Interaction Study

To determine which webpage interactions are considered "useful" by users, we conducted a user interaction study. A group of participants was recruited to browse a curated set of websites randomly selected from a list of popular domains. During each session, participants were asked to interact with the websites as they normally would. With their consent, their sessions were recorded to support the evaluation and training of the computer vision model.

Each participant was tasked with identifying:

(1) the interactive elements present on the webpages that they are more likely to use, and
(2) the nature of their interactions with those elements.

Participants initially interacted with a designated set of websites as they would normally do, some of which were repeated across sessions for consistency and cross-validation. We specified the instruction that we did not want them to find all interactions in the webpage (9). Following the interaction phase, participants were asked to annotate screenshots of the websites. They labeled each interactive element according to its purpose, such as buttons, hyperlinks, and input fields. These annotated screenshots were then processed into structured objects, containing bounding box coordinates and corresponding labels, for input into the machine learning

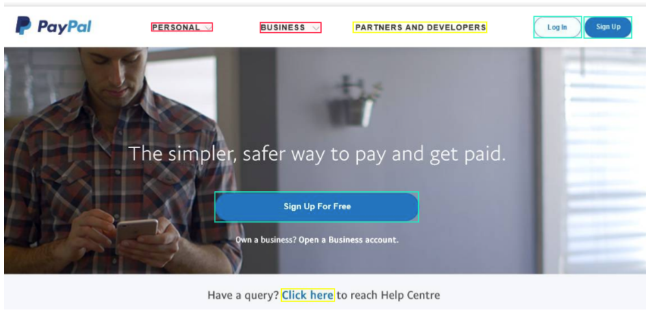

PayPal
PERSONAL
BUSINESS
PARTNERS AND DEVELOPERS
Log In
Sign Up
The simpler, safer way to pay and get paid.
Sign Up For Free
Own a business? Open a Business account.
Have a query? Click here to reach Help Centre

Figure 1: Participant's Label Example

model. Throughout the study, video recordings of the participants' actions (collected with consent) were used to correlate their actual behaviors with the labeled elements.

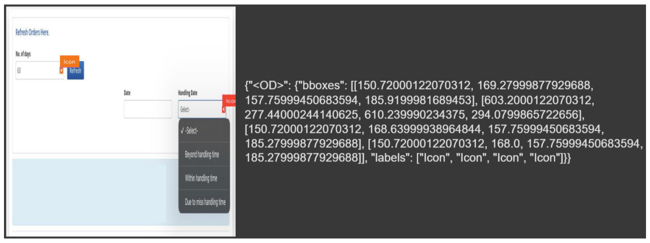
3.2 Interactive Elements Identification

Since we are deploying a black-box approach for code elimination, it is inherently difficult to determine the full set of interactive elements present on a webpage. To enable the automated identification of these elements, we fine-tuned Florence 2, a computer vision model developed by Microsoft, using a dataset derived from both participant-labeled interactions and publicly available interactive element annotations on Roboflow [1].

A preliminary experiment using a short training duration of 10 epochs yielded promising results, particularly in the accurate identification of dropdown menus and buttons. Figure 2 shows an example of the output from Florence 2 after fine-tuning. The model outputs the detected interactive elements on a webpage based on the labeled training data. Florence 2 is then evaluated against a publicly available computer vision model trained solely on Roboflow to benchmark its performance in identifying interactive elements.

To integrate the computer vision model with Muzeel, we capture a screenshot of each webpage after it is fully loaded. Screenshots from sites that fail to load or are blocked by third-party providers are filtered out. The valid screenshots are passed through the model, which produces a list of interactive elements detected in the visual representation of the page. It is important to note that, because this approach relies solely on the initial page load screenshot, any hidden or dynamically revealed interactions (e.g., dropdowns revealed on hover) may not be included in the image and thus will not be recognized by the model. However, we treat these interactions as valid targets for elimination, based on the assumption that they are not immediately accessible to users during typical browsing sessions.

The final output of this process is a deduplicated list of interactive elements identified on the page. This provides a preliminary understanding of the interactive landscape of



```
{"<OD>": {"bboxes": [[150.72000122070312, 169.27999877929688,
157.75999450683594, 185.9199981689453], [603.2000122070312,
277.44000244140625, 610.239990234375, 294.0799865722656],
[150.72000122070312, 168.63999938964844, 157.75999450683594,
185.27999877929688], [150.72000122070312, 168.0, 157.75999450683594,
185.27999877929688]], "labels": ["Icon", "Icon", "Icon", "Icon"]}}
```

Figure 2: Example Output of Florence-2

the website, which serves as a foundation for subsequent filtering and optimization steps.

3.3 Interaction Identification

Given our familiarity with standard web design patterns, users often expect certain default interactions for specific website elements. However, identifying interactions alone is not sufficient for filtering out infrequently used ones, as each element may support multiple interactions, some of which are rarely triggered. To address this, after obtaining the list of DOM elements on a webpage through the computer vision model, we process this list using a large language model (GPT-3.5-turbo).

Specifically, we provide GPT-3.5-turbo with a predefined list of default interactions as part of the prompt, instructing it to return only those interactions [8]. This constraint is crucial, as it ensures consistency with Muzeel's legacy implementation, which relies on specific interaction keywords for code elimination. For example, a double-click interaction might be described as either double-click or dbclick, but only the latter is recognized by Muzeel. By explicitly specifying acceptable terms in the prompt, we ensure that only compatible interactions are returned.

The model is further instructed to list only common and likely interactions, rather than exhaustively listing every potential behavior associated with an element. The final output is a non-repetitive list of standardized interaction labels corresponding to each website, which can then be used for subsequent filtering and analysis.

Listing 1: Example User Prompt for Interaction Prediction

```
user_prompt = {
    "role": "user",
    "content": (
        f"You are being given a list of HTML
             elements found on a website: [{
           elements_str}] "
        "Your task is to identify the list
            of user interaction event types
            COMMONLY associated with these
            elements. DO NOT TRY TO FIND ALL
             INTERACTIONS."
        "Only return a single flat list of
            interactions     do not group by
             element or give explanations.\n
            \n"
        "ONLY include interactions from this
             fixed set: "
        "click, mousedown, mouseup, focus,
            blur, mouseover, mouseenter,
            mouseout, mouseleave, "
        "keydown, keypress, keyup, input,
            change, dblclick, drag,
            dragstart, dragend.\n\n"
        "Format your response exactly like
            this (nothing more):\n"
        "<response>[click, focus, mouseover
            ]</response>\n\n"
        "Do not include JSON, element labels
            , or any other formatting."
    )
}
```

3.4 Element Filtering

After identifying all interactive elements and their corresponding interactions on a webpage, we leverage Muzeel's existing mechanism to perform interaction-based filtering. Specifically, Muzeel includes an interaction bot that automatically navigates through a webpage and simulates user interactions with all elements, including nested (child) elements.

The bot operates using an XPath tree structure, beginning with the root nodes (e.g., <html> or <body>) and traversing downward to interact with child nodes. In our implementation, before the bot initiates any interactions, we filter its target XPaths using the interactions and elements generated from our models. As a result, the bot proceeds only with interactions on the elements previously identified as commonly interactive elements.

Additionally, the bot also interacts with any child elements of the identified parent elements, as these children are likely to be interacted with during typical user behavior. At the end of this process, the bot produces a log containing the elements it interacted with which are the elements considered "useful" for the user experience. Alternatively, any elements or interactions not activated by the bot are our targeting codes for elimination. These represent either the "dead code" as defined in Muzeel or the "infrequently used code" targeted by VMuzeel.

3.5 Code Elimination

VMuzeel utilizes the same code elimination mechanism implemented in Muzeel. Once the interaction bot returns a log of elements it has interacted with, which are the useful or active elements, we maintain Muzeel's mechanism to identify and eliminate any functions not triggered during the interaction process. However, in VMuzeel, this elimination is applied on top of the Muzeeled version of the website, which has already had its dead code removed. In other words, VMuzeel performs an additional step of code elimination targeting infrequently used interactions, thereby further optimizing an already cleaned website. The resulting files are saved with a .v extension to distinguish them from the original Muzeel output. This step does not require any modification to Muzeel's core logic, as VMuzeel builds directly upon its original implementation.

3.6 VMuzeel Evaluation

To evaluate the effectiveness of VMuzeel, we deploy an automated agent to collect performance metrics such as page time, Speed Index, and other relevant statistical indicators. These measurements are gathered from three sets of websites: the original (unaltered) websites, the Muzeeled versions, and the VMuzeeled versions. By comparing these datasets, we aim to assess whether VMuzeel provides performance improvements beyond Muzeel and to evaluate whether further code elimination compromises the functionality or usability of the original websites.

4 EVALUATION

To evaluate the effectiveness of VMuzeel, we tested it against 100 randomly selected websites from the Tranco list and compared the results with those of their Muzeeled counterparts. Each website was tested five times to account for potential anomalies caused by fluctuations in network conditions. All load evaluations were conducted using the Chrome browser on an Android device to ensure consistency across tests for both Muzeel and VMuzeel versions.

4.1 Page Load Time

Figures [3] and [4] present the Cumulative Distribution Function (CDF) of load times and fully loaded times across three versions of each website: the original, Muzeel (with dead code removed), and VMuzeel (further optimized).

As shown in the CDF graphs, the VMuzeel version consistently shifts to the left of both the original and Muzeel curves, indicating faster load times overall. Notably, VMuzeel shows a significant improvement for pages that load in under 60,000 milliseconds, highlighting its effectiveness in accelerating page performance, especially under typical network conditions. Additionally, the CDF for fully loaded time (Figure [4])

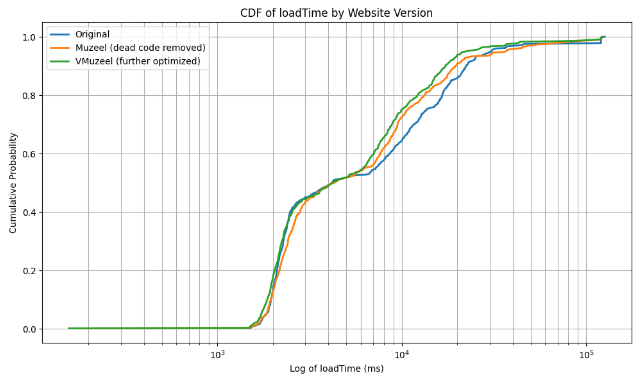


Figure 3: CDF of loadTime

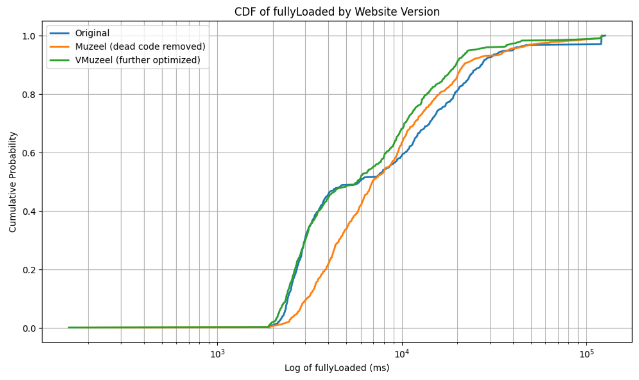


Figure 4: CDF of fullyLoadTime

demonstrates a larger performance gain with VMuzeel than with Muzeel alone. This suggests that VMuzeel's further elimination of infrequently used interactions has a measurable positive impact, particularly in environments with slower or less stable connections.

Figure [5] illustrates the percentage improvement in load time across 10 randomly selected websites when comparing VMuzeel with Muzeel. The performance gains range from 2% to over 30%. Websites with minimal improvement likely already have a high percentage of their interactions correctly preserved in Muzeel, resulting in fewer further optimizations by VMuzeel. However, because the degree of interactivity varies significantly between websites, the number of infrequently used elements also differs, accounting for the variance in observed improvements.

4.2 File Size Comparison

As VMuzeel performs elimination on the Muzeel-generated websites, we expect a further reduction in the size of cloned files within the VMuzeel output. Figure [6] illustrates the difference in file sizes for files with the same names between

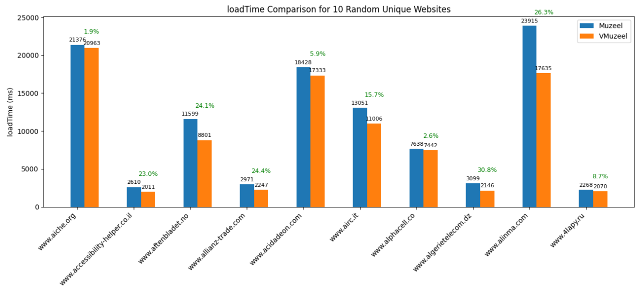


Figure 5: Load Time Comparison of 10 Random Websites

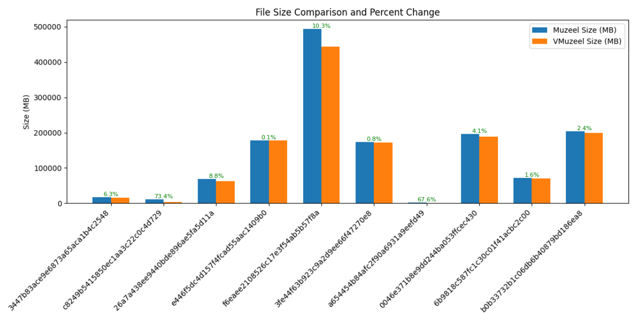
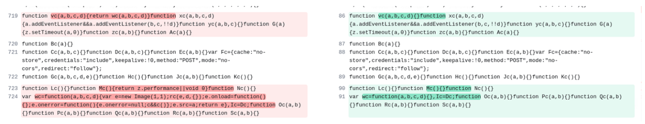


Figure 6: Muzeel vs VMuzeel File Size



```
function vc(a,b,c,d){return wc(a,b,c,d)}function xc(a,b,c,d)
{a.addEventListener&&a.addEventListener(b,c,!!d)}function yc(a,b,c){}function G(a)
{z.setTimeout(a,0)}function zc(a,b){}function Ac(a){}
function Bc(a){}
function Cc(a,b,c){}function Dc(a,b,c){}function Ec(a,b){}var Fc={cache:"no-
store",credentials:"include",keepalive:!0,method:"POST",mode:"no-
cors",redirect:"follow"};
function Gc(a,b,c,d,e){}function Hc(){}function Jc(a,b){}function Kc(){}
function Lc(){}function Mc(){return z.performance||void 0}function Nc(){}
var wc=function(a,b,c,d){var e=new Image(1,1);rc(e,d,{});e.onload=function()
{};e.onerror=function(){e.onerror=null;c&&c()};e.src=a;return e},Ic=Dc;function Oc(a,b)
{}function Pc(a,b){}function Qc(a,b){}function Rc(a,b){}function Sc(a,b){}
```

```
function vc(a,b,c,d){}function xc(a,b,c,d)
{a.addEventListener&&a.addEventListener(b,c,!!d)}function yc(a,b,c){}function G(a)
{z.setTimeout(a,0)}function zc(a,b){}function Ac(a){}
function Bc(a){}
function Cc(a,b,c){}function Dc(a,b,c){}function Ec(a,b){}var Fc={cache:"no-
store",credentials:"include",keepalive:!0,method:"POST",mode:"no-
cors",redirect:"follow"};
function Gc(a,b,c,d,e){}function Hc(){}function Jc(a,b){}function Kc(){}
function Lc(){}function Mc(){}function Nc(){}
var wc=function(a,b,c,d){},Ic=Dc;function Oc(a,b){}function Pc(a,b){}function Qc(a,b)
{}function Rc(a,b){}function Sc(a,b){}
```

Figure 7: Example of File Differences Between Muzeel (left) and VMuzeel (right)

Muzeel and VMuzeel. Our analysis focuses only on files that exist in both versions, as some Muzeel files do not require elimination and therefore remain unchanged, with no corresponding VMuzeel file generated.

We observe that the differences in file sizes vary across files, but the overall trend shows that VMuzeel files are generally smaller due to the additional elimination steps. The Figure [7] demosntrates how the difference in file between Muzeeled website and VMuzeeled website. This variation in elimination effectiveness is largely attributed to differences in website file structures. Each website may organize functions related to interactive elements differently, which influences how much can be eliminated. Files that show significant reductions in size, and higher elimination rates—are more likely to contain hidden interactive elements.

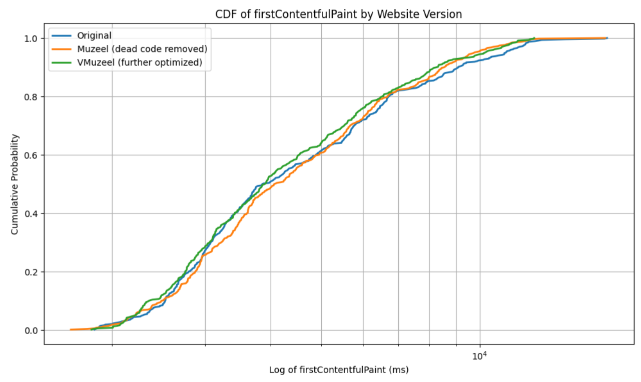


Figure 8: CDF of firstContentfulPaint

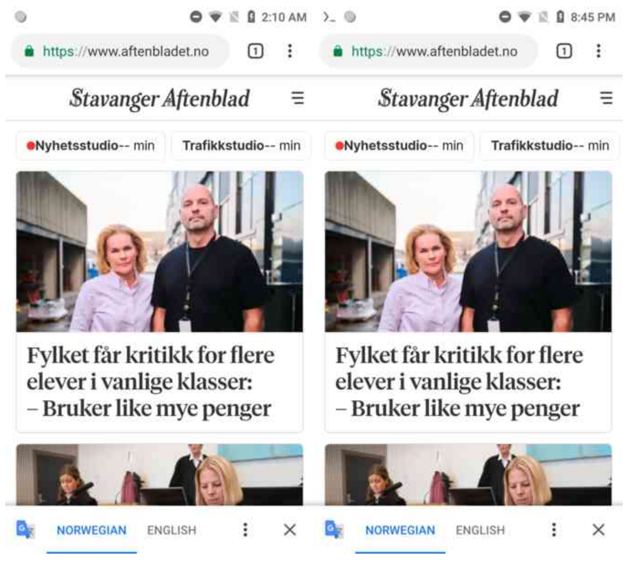

https://www.aftenbladet.no
Stavanger Aftenblad
Nyhetsstudio-- min
Trafikkstudio-- min
Fylket får kritikk for flere elever i vanlige klasser: – Bruker like mye penger
NORWEGIAN
ENGLISH

Figure 9: Website visual between Muzeel (left) and VMuzeel (right)

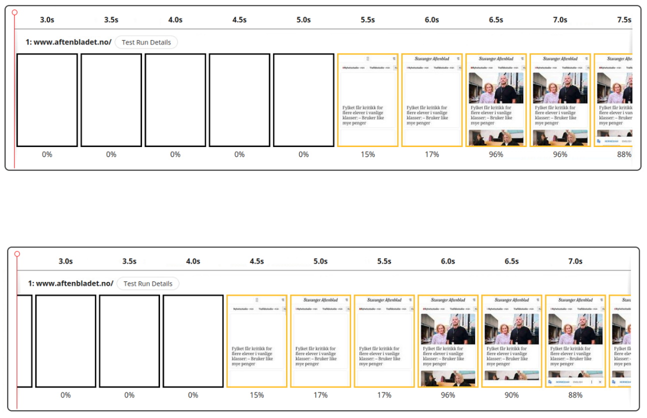


Figure 10: Visual Time Frame of Muzeeled (top) VMuzeeled (bottom) Website

4.3 Website Visual Comparison

The graph in Figure [8] displays the First Contentful Paint (FCP), a key web performance metric that captures the moment when the first visual element is rendered on the screen. This metric often shapes a user's perception of a website's speed. The graph shows a leftward shift for VMuzeel compared to both the Muzeeled and original websites, indicating faster perceived load times.

As noted in the original Muzeel paper, the FCP values for Muzeel and VMuzeel are expected to be similar since JavaScript files are typically loaded after this point [4]. However, we still observe a slight improvement in FCP for VMuzeel in the lower time range, suggesting some marginal gains in visual responsiveness. To assess the impact of VMuzeel on visual fidelity, we randomly selected five websites and compared their appearances after applying Muzeel and VMuzeel. Example screenshots show that VMuzeel does not compromise the visual appearance of the websites. This is because VMuzeel targets only the JavaScript functions responsible for interactive elements, leaving the onload functions—and therefore the initial visual structure—intact. The filtering and elimination occur after the loading phase, preserving the website's original look.

Figure [10] presents two timelines comparing the loading sequences of Muzeeled and VMuzeeled websites. The results show that VMuzeeled websites follow the same visual loading steps but at a faster pace. The visuals appear almost identical, just delivered more quickly. However, while the visual similarity and performance improvements are promising, further evaluation is needed to fully assess any impact on website functionality.

5 LIMITATION

5.1 Limited Information from Screenshot

A screenshot of the website is captured after it has fully loaded, containing all the elements visible to the user at that moment. However, certain dynamic elements and interactive behaviors cannot be accurately captured in a single screenshot. For instance, videos are treated as static images, meaning that any interactive features, such as play buttons or video controls, may be misidentified or ignored entirely.

Moreover, since the current approach uses only one screenshot, it fails to capture content that appears after user interactions such as hovering or clicking. Examples include pop-up sections revealed on hover, dropdown menus, or initial modal dialogs that block the main content until dismissed. These scenarios prevent the screenshot from accurately representing the full range of content and interactions available on the page, leading to incomplete data and potentially incorrect elimination decisions.

Although we have attempted to mitigate this limitation by enabling the bot to interact with any child elements under the identified interactive components, there is still a risk of missing elements. Specifically, if certain elements are dynamically added to the DOM only after user interaction, they may be excluded from the elimination process, resulting in incomplete or inaccurate filtering.

5.2 Model Hallucinations and Over Estimation

Since our focus is on identifying commonly used user interactions, we intentionally avoid having the computer vision model detect all elements on a website. Doing so would contradict our goal of targeting infrequently used or redundant code. However, accurately distinguishing between frequently and infrequently used elements remains challenging due to the limited amount of training data and the diverse nature of UI/UX design across websites. Currently, we attempt to address this limitation by adjusting hyperparameters during fine-tuning, rather than expanding the dataset. While this helps to a certain extent, it is not a complete solution and may still lead to incorrect prioritization or omission of interactive elements.

In addition to computer vision, we use a large language model (GPT-3.5) to infer possible interactions between elements. This model must follow a strict output format to ensure compatibility with Muzeel's filtering system. If the language model deviates from the expected format, it can disrupt the parsing process and compromise the overall pipeline. For example, if the model outputs "doubleclick" instead of the expected "dbclick," Muzeel will fail to recognize the interaction, leading to incorrect or incomplete filtering.

These types of hallucinations and formatting errors highlight the fragility of the current system and the importance of robust validation mechanisms to ensure consistency and accuracy in interaction identification.

5.3 Incomplete Understanding of Website Structures

A significant challenge in the filtering process is the difficulty in accurately linking the XPath of an element to output generated by the computer vision model. Many websites use styling libraries such as Bootstrap or TailwindCSS, which often result in non-descriptive or generic element IDs. This lack of semantic clarity makes it challenging to ensure that the elements identified by the model correspond precisely to the interactive elements being targeted by the bot.

The current implementation relies on both computer vision and a large language model to determine which elements should be filtered. However, since the model output often refers to general element types (e.g., "bg-white flex y-4") rather than specific instances, the filtering process can lack precision. As a result, some elements may not be filtered as strictly or accurately as intended, leading to either missed interactions or unintended eliminations.

6 FUTURE WORKS

6.1 Series Capturing of the Website

As discussed in the limitations section, evaluating a website using a single screenshot offers only a preliminary understanding of its interactive elements and fails to capture the full range of user interactions over time. To address this, future work can involve capturing a series of screenshots at different stages of the user's interaction with the website. This sequential approach would provide a more comprehensive view of dynamic content and element behavior. For example, the model can further predict the possible next state of the website, which may include additional DOM elements or new interaction states.

By incorporating multiple snapshots from different interaction states, we can improve the accuracy of both the computer vision and large language models. Specifically, this method can help reduce the misidentification of elements that appear only after certain user actions, such as clicks or hovers. This increase in dataset would enhance model training and ultimately lead to more precise interaction filtering and elimination.

6.2 Alternative Approach to Model Fine-Tuning

Currently, due to the limited availability of labeled datasets, we use two separate models: one for detecting elements on a webpage and another for predicting the interactions associated with those elements. However, a more effective approach would involve using a unified dataset in which each element is directly labeled with its corresponding interactions. For instance, a button element could be labeled with interactions like click, mousedown, and mouseup, while an input field could be labeled with keydown and keyup. This would allow the model to learn the relationship between visual context and interaction type more accurately, resulting in more precise and efficient interaction prediction. Furthermore, with such a dataset, we could eliminate the need for a separate large language model, which currently lacks visual context when predicting interactions.

6.3 Improved Structural Understanding of Website

As noted in the limitations section, we currently lack a complete understanding of how elements are structured and identified within a website. Gaining insight into foundational aspects—such as whether the website uses styling libraries like TailwindCSS or Bootstrap—could help us better interpret how element IDs are generated and structured. With this information, instead of relying solely on computer vision to identify DOM elements, we could fine-tune the model to take into account styling conventions when making predictions. For example, if the model is aware that a website uses TailwindCSS, it could better interpret the meaning and patterns of class names and IDs.

Ultimately, VMuzeel could be enhanced to extract and analyze metadata from the website headers to detect which libraries or frameworks are in use. This additional context would enable the model to generate predictions that are more accurately aligned with the actual DOM structure and element identifiers, leading to more reliable filtering and interaction mapping.

7 CONCLUSION

We enhanced Muzeel by integrating a computer vision model and a large language model to better predict website behavior. Our process began with a user interaction study, conducted with participant consent, to collect preliminary data on typical user behaviors when interacting with websites. This data was then used to fine-tune a computer vision model capable of identifying useful interactive elements present on a webpage. These identified elements were further processed through a large language model to determine additional possible interactions. The combined output served as the basis for filtering out unutilized interactions in Muzeel. Elements that were not interacted with, and therefore not logged into Muzeel, were considered “dead code” for removal.

This optimization, referred to as VMuzeel, demonstrated positive outcomes in terms of load time reduction and file size savings. Compared to the original Muzeel, VMuzeel achieved faster load times across the same set of websites. While the degree of file size reduction varied due to differences in how elimination was distributed across files, overall performance was improved. Significantly, the visual appearance of the webpages remained unchanged from the Muzeeled versions, as filtering was applied only after all elements had fully loaded.

Despite these improvements, several limitations remain. Accurately filtering infrequently used interactions still poses challenges, particularly in ensuring that the machine learning models generate precise and reliable output. Future work will focus on addressing these challenges by refining the models and enhancing the relevance and accuracy of their predictions in the context of real-world webpages.

REFERENCES

[1] Wire frame. 2024. all-elements Dataset. https://universe.roboflow.com/wire-frame/all-elements-l5mud. https://universe.roboflow.com/wire-frame/all-elements-l5mud visited on 2024-12-14.

[2] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. 2017. Improving user perceived page load times using gaze. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 545–559.

[3] Jesutofunmi Kupoluyi, Moumena Chaqfeh, Matteo Varvello, Russell Coke, Waleed Hashmi, Lakshmi Subramanian, and Yasir Zaki. 2022. Muzeel: Assessing the impact of javascript dead code elimination on mobile web performance. In *Proceedings of the 22nd ACM Internet Measurement Conference*. 335–348.

[4] Xuanzhe Liu, Jinfeng Wen, Zhenpeng Chen, Ding Li, Junkai Chen, Yi Liu, Haoyu Wang, and Xin Jin. 2023. FaaSLight: General Application-level Cold-start Latency Optimization for Function-as-a-Service in Serverless Computing. *ACM Trans. Softw. Eng. Methodol.* 32, 5, Article 119 (jul 2023), 29 pages. https://doi.org/10.1145/3585007

[5] Ivano Malavolta, Kishan Nirghin, Gian Luca Scoccia, Simone Romano, Salvatore Lombardi, Giuseppe Scanniello, and Patricia Lago. 2023. Javascript dead code identification, elimination, and empirical assessment. *IEEE Transactions on Software Engineering* (2023).

[6] Usama Naseer, Theophilus A Benson, and Ravi Netravali. 2021. Webmedic: Disentangling the memory-functionality tension for the next billion mobile web users. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*. 71–77.

[7] Ravi Netravali, Vikram Nathan, James Mickens, and Hari Balakrishnan. 2018. Vesper: Measuring {Time-to-Interactivity} for Web Pages. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 217–231.

[8] OpenAI. 2025. GPT-3.5-Turbo. https://platform.openai.com/docs/models/gpt-3.5-turbo

[9] Piotr Skalski. 2024. Florence-2: Open Source Vision Foundation Model by Microsoft. https://blog.roboflow.com/florence-2/