EMSE manuscript No.
(will be inserted by the editor)

Web Element Relocalization in Evolving Web Applications: A Comparative Analysis and Extension Study

Anton Kluge · Andrea Stocco

the date of receipt and acceptance should be inserted later

Abstract Fragile web tests, primarily caused by locator breakages, are a persistent challenge in web development. Hence, researchers have proposed techniques for web-element re-identification in which algorithms utilize a range of element properties to relocate elements on updated versions of websites based on similarity scoring. In this paper, we replicate the original studies of the most recent propositions in the literature, namely the Similo algorithm and its successor, VON Similo. We also acknowledge and reconsider assumptions related to threats to validity in the original studies, which prompted additional analysis and the development of mitigation techniques. Our analysis revealed that VON Similo, despite its novel approach, tends to produce more false positives than Similo. We mitigated these issues through algorithmic refinements and optimization algorithms that enhance parameters and comparison methods across all Similo variants, improving the accuracy of Similo on its original benchmark by 5.62%. Moreover, we extend the replicated studies by proposing a larger evaluation benchmark (23× bigger than the original study) as well as a novel approach that combines the strengths of both Similo and VON Similo, called HybridSimilo. The combined approach achieved a gain comparable to the improved Similo alone. Results on the extended benchmark show that HybridSimilo locates 98.8% of elements with broken locators in our testing scenarios.

Keywords Web Testing; Locators; Test Robustness; Test Maintenance; Test Evolution

A. Kluge
Technical University of Munich – Boltzmannstraße 3 Garching near Munich, Germany
E-mail: anton.kluge@tum.de
A. Stocco
Technical University of Munich – Boltzmannstraße 3 Garching near Munich, Germany and fortiss GmbH – Guerickestraße 25 Munich, Germany
E-mail: andrea.stocco@tum.de—stocco@fortiss.org

1 Introduction

Automated end-to-end (E2E) web tests created with tools such as Selenium are renowned for being fragile as the the web application under test evolves [14, 20]. Researchers have singled out web element *locators* as the main cause of fragility [10, 36]. Locators are commands used by test automation tools to identify elements on a web page, hanging on specific properties found in the Document Object Model (DOM), such as the element's identifier, XPath, or text.

Locator breakages are mainly caused by code changes in the web application. Given the short release cycles and advancements in modern websites, such breakages occur frequently. This poses a significant challenge for automated testing, as it requires manual fixing of tests before they can be executed again. This process is time-consuming and frustrating for testers. As a result, test suites are often abandoned [4], as the effort outweighs the benefits.

To mitigate these issues and minimize the number of fragile tests, researchers have proposed automated algorithms that produce robust locators [12, 19, 20, 21, 23, 24, 25]. The current state-of-the-art approach for locating a web element corresponding to a specific locator is the Similo algorithm [24]. This algorithm calculates a similarity score based on multiple properties of web elements to re-identify the target element among a set of candidates, which consist of all elements on an updated version of the website. The algorithm selects the candidate with the highest similarity score as the new target element. The original algorithm has been extended in two follow-up works, namely VON Similo [23] and LLM VON Similo [25]. The former extends the original Similo algorithm by comparing groups of visually overlapping elements on a website instead of singular elements. The latter leverages a large language model to improve the algorithm's accuracy.

We chose to focus on the Similo algorithm [24] and its extensions because recent systematic literature reviews on web application testing [2] identify Similo as the current state-of-the-art approach for repairing broken web test cases. Similo has shown superior performance compared to baseline methods such as Robula+, Vista, and WATER. Additionally, locator-based techniques (e.g., Selenium) have been reported to outperform purely visual techniques in terms of robustness and reliability [34]. The Similo algorithm is conceptually similar to other locator-identification algorithms like COLOR [12], which have demonstrated viability. Moreover, VON Similo and LLM VON Similo, both extensions of Similo, offer novel approaches (e.g., visually overlapping nodes, integration of language models) which motivated us to replicate and further investigate their effectiveness. During our initial review of Similo and its extensions, we observed several limitations and threats to validity in the original evaluations. Initial theories on how to address these limitations led us to extensively re-assess and extend the original results in order to ensure their robustness and applicability.

First, the original Similo algorithm relies on a fixed set of web element properties and weights to web element re-localization. Second, the benchmark used to evaluate Similo contains web elements from versions 12 to 60 months apart. This range does not accurately reflect the actual update frequency of websites, nor it aligns with continuous integration environments, where updates tend to be smaller, more regular, and tests are conducted frequently. Third, we found a significant revision of the evaluation benchmark between the original study and its

subsequent extensions, rather than also assessing the extensions using the initial benchmark for a consistent comparison.

Motivated by the will to understand the causes of these discrepancies and to address the aforementioned challenges, in this paper, we replicate the Similo [24] and VON Similo [23] studies, improving the experimental setting of the original papers to address the identified limitations and threats to the validity. More in detail: (1) we improve Similo by optimizing the attributes and weights of the original algorithm. We evaluated six *new* similarity functions to calculate the similarity between web element properties and optimize the weights assigned to these similarities using a genetic algorithm. We also analyze the capabilities of a novel hybrid version that combines the capabilities of Similo and VON Similo. (2) We collected a benchmark dataset of more than 10,000 element pairs from multiple websites and versions over the past five years. Our benchmark is 12× bigger than the original Similo benchmark and 23× bigger than the VON Similo benchmark, and it is more reflective of the fine-grained modifications occurring in real web sites. (3) We perform a fairer comparison between the original Similo and its extensions by evaluating all algorithms on the same benchmarks, both using the original as well as our new extended benchmark. While we were able to replicate the original study results, our findings are in contrast with ones by Nass et al. [23] as we show that VON Similo under-performs Similo in directly identifying the target element but excels in identifying the visual overlap of the target element.

Our paper makes the following contributions:

- **Replication.** A replication study of the results of the Similo and VON Similo algorithms, including a comparison with the experimental setup and benchmarks used in the Similo and LLM VON Similo algorithms. Ours is the first attempt at evaluating all Similo and VON Similo algorithms on the same benchmark under analogous conditions.
- **Extended Benchmark and Metrics.** We extended the original benchmark of 804 element pairs to 10,376 element pairs and introduced six additional metrics.
- **Library.** A library for the Selenium framework, which is publicly available [29]. Our tool wraps an existing locator and uses our extended Similo algorithm to locate an element if the original locator fails.

2 Motivating Example

In this section, we describe the problems occurring to E2E web tests during web app evolution. We use as a running example the home page of Zoom.us, a popular online chat service used for video communications, messaging, voice calls, conference rooms for video meetings, and virtual events. Figure 1 (top) shows the website on September 2022, whereas Figure 1 (bottom) shows the update website on January 2023. In the short timespan of four months, the website has undergone a significant redesign, which has affected the locators of the elements and likely broke possible test cases. In addition to stylistic changes, some buttons and links were relocated in the GUI (e.g., the Host drop-down list). In the rest of this section, we describe the variety of breakage scenarios that can occur in web tests, which robust relocalization techniques should aim to handle [36].

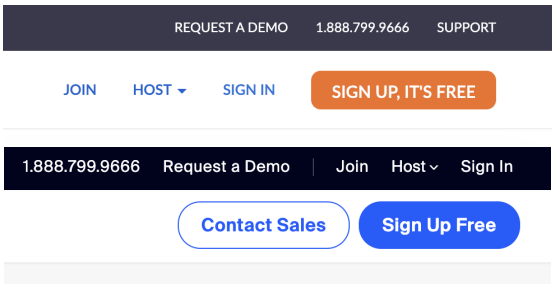

Fig. 1: Zoom.us homepage on September 2022 (top) and January 2023 (bottom).

2.1 Element Not Found

When the provided locator fails to identify an element on an updated website, the test case will break [10, 36]. This requires a developer to manually locate the element on the updated website and modify the locator in the test case.

Automated techniques such as Similo aim to correctly identify the element on the updated version of the website. The algorithm requires a working reference web app, typically an old version of the website in which the web test used to function, to gather a set of properties. These properties are then used to try find the element on the new version of the website. For instance, the "Request a Demo" button has the same tag and non-capitalized text, and a similar shape, location, and neighboring text. All such information are used by Similo in correctly identifying the link on the updated website. By comparing the properties of the old element with the properties of all elements on the new website, Similo returns the most similar element, which likely identifies the original web element.

2.2 False Positive

Another common reason for test breakages occurs when the locator returns another existing element instead of the intended target [36]. Repairing these breakages is particularly challenging because the developer has to manually trace where the test deviates from its intended path. For instance, the button initiating a direct call to 1.888.799.9666 in Figure 1 (top) can be identified by the CSS Selector `#black-topbar > div > ul > li:nth-child(2) > a` [22]. The same locator will return the "Support" button in Figure 1 (bottom), even though they serve different purposes. A test case will not break immediately, as the locator returns an existing element, but later on after the test case execution has deviated from the intended path. Similo can help identify these false positives by validating if another element on the updated website.

In case of correct detection, Similo could actually support the automated repair of the broken locators. While Similo does not store the information required to locate the element in the source code, as the amount of data collected would clutter the test case, it could in practice use some form of caching mechanism to store the information from the old version of the website. By saving an identifiers for each

web element in cache, Similo could update the information associated with such web elements on new evolved DOM versions.

2.3 Misclassifications

The internal functioning of Similo may cause the algorithm to misclassify elements. One reason is due to the target element changing its tag. As the tag property is highly weighted in Similo calculations, other elements in the close proximity of the target which have the same tag as the target element can be misclassified. For example, the "Join" button in Figure 1 (top) has the tag `a`, and the tag `button` in Figure 1 (bottom). Because the "Sign Up Free" button in Figure 1 (bottom) has the same tag as the target element, as well as similar shape, location, neighboring text and XPath, Similo misclassifies the "Sign Up Free" button as the "Join" button. This is especially problematic as Similo always returns an element. If the element is not the intended target, the test case will break at an arbitrary point in the test execution, making it harder for developers to identify the root cause of the breakage.

3 Approaches

This paper is a replication and extension of the work by Nass et al. [24] presented in the ACM Transactions on Software Engineering and Methodology (vol. 32, no. 3) in 2023.

The algorithm is based on the idea that when a web element is modified, some properties are altered while others remain the same or undergo minor changes. Thus, the main working assumption is that by calculating a similarity score between two elements, the algorithm can identify the element with the highest similarity score as the target element. Two successors to the basic Similo algorithm have been developed: VON Similo [23] and LLM VON Similo [25]. The first successor, VON Similo, takes advantage of the fact that visual entities on a website often consist of multiple concrete web elements, which can help to identify the target element. The second successor (LLM Von Similo) utilizes a Large Language Model (LLM) to locate the correct element from a pre-selected set of candidates.

For all Similo variants, the algorithm starts with an element that can be reliably identified on an old, baseline version of a specific website. This element is referred to as the target. The algorithm extracts all necessary information about this element. Then, the algorithm tries to find the target element among all candidates elements on a new version of the same website, where the locator used to identify the target element on the baseline version no longer works. The algorithm calculates a similarity score between the target element and all candidates. The candidate with the highest similarity score is then returned as the target element. It is important to note that Similo is not a locator algorithm. It does not generate a robust locator for the target element, but rather identifies the target element among all candidates. A robust locator for the target element must be generated using other methods, such as Robula+ [20], COLOR [12], or the MultiLocator [19].

In the following sections, we describe each algorithms in a greater level of detail, using the following nomenclature. E will always refer to an arbitrary web

Algorithm 1 Similo

```
1: function SIMILO(target_element, candidate_elements)
2:     best_score ← 0
3:     best_candidate ← null
4:     for all candidate ∈ candidate_elements do
5:         score ← SIMILOSCORE(target_element, candidate)
6:         if score > best_score then
7:             best_score ← score
8:             best_candidate ← candidate
9:         end if
10:    end for
11:    return best_candidate
12: end function
13: function SIMILOSCORE(target, candidate)
14:    total ← 0
15:    for all property ∈ target.properties do
16:        t ← GET(target, property)
17:        c ← GET(candidate, property)
18:        sim ← SIMILARITY(t, c)
19:        total ← total + sim × WEIGHTS[property]
20:    end for
21:    return total
22: end function
```

element. T will be used for target elements, the original version of an element that needs to be found on an updated website, and C for a possible candidate for the target on the updated website. The candidate on the new version of a website that corresponds to the target element will be noted as C'. If multiple elements are present, they will be referred to as $E_1, .., E_n$. When working with properties, $E_n.a_m$ refers to the m-th property of the n-th element. A visual overlap for the element E is noted as O^E.

3.1 Similo

Similo (Algorithm 1) requires a working version of the web application where the web element can be identified. Given the original web element T and its properties $T.a_1 ..., T.a_n$, as well as all new web elements $C_1, ..., C_n$, of which one should be the original one C', Similo computes a similarity score between T and each $C_1, ..., C_n$ individually:

$$\text{Similo}(T, C) = \sum_{i \in \#\text{properties}} \text{similarity}(C.a_i, T.a_i) \cdot c_i$$

Here, c_i is a property-specific weight based on the COLOR study [12], determining how effective, stable, and unique specific properties are. Stable properties, such as tag and name, are assigned a weight of 1.5, while non-stable properties receive a weight of 0.5. The similarity function calculates a score ranging from $[0, 1]$ on how similar both values are using a predetermined algorithm. This score is used to generate a ranking of all web elements based on their similarity with the original element, enabling developers to choose the element with the highest similarity score or retain all elements above a certain threshold.

The specific properties used in the Similo algorithm can be found in Table 1. These properties are selected based on the locator types supported by the Selenium WebDriver API (id, name, class, tag, link text, partial link text, XPath, and

Table 1: Properties, similarity functions, and weights used by Similo [24].

	Similarity Function	**Weight**	**Description**
Tag	Equality	1.5	Element type
Class	Levenshtein Similarity	0.5	CSS class
Name	Equality	1.5	Attribute value
ID	Equality	1.5	Unique identifier
HRef	Levenshtein Similarity	0.5	Link address
Alt	Levenshtein Similarity	0.5	Image text
Absolute XPath	Levenshtein Similarity	0.5	Full DOM path
ID-XPath	Levenshtein Similarity	0.5	Relative DOM path
Is Button	Equality	0.5	`button` tag or `a` tag with `btn` in class or a `input` with button, sumbit or reset type.
Location	Euclidean Distance	0.5	X and Y coordinates
Area	Euclidean Distance	0.5	Calculated as width * height
Shape	Euclidean Distance	0.5	Calculated as width / height
Visible Text	Levenshtein Distance	1.5	Value, inner text or placeholder
Neighbor Texts	Word set similarity	1.5	All visible text in a larger rectangle around the element

CSS) for native element location. The authors also consider the locators chosen by Selenium idE (a tool for recording and replaying user interactions), including id, link text, name, and various XPaths. Additional properties are chosen based on the results of the COLOR study (id, class, name, value, type, tag name, alt, src, href, size, onclick, height, width, XPath, X-axis, Y-axis, link text, label, and image) and the WATER study [3] (id, XPath, class, link text, name, tag, coord, clickable, visible, z-index, hash). The authors selected all DOM-based properties from this list and excluded properties extracted from the visual user interface. Properties with slight differences between versions, such as class, links, and XPaths, are compared using the Levenshtein distance. On the other hand, properties that tend to change completely, such as tag and id, are compared using equality. Integer-based properties are compared using the Euclidean distance.

3.2 VON Similo

In this paper we also replicate and extend VON Similo, an extension of the basic Similo algorithm introduced by Nass et al. [23] and presented in the IEEE Conference on Software Testing, Verification and Validation (ICST) in 2023 [23].

Algorithm 2 Von Similo

Require: original element T, old web page $OldSite$, candidate elements C
Ensure: best matching element in C

function VonSimilo($T, OldSite, C$)
  $T\_overlap \leftarrow$ Overlap($T, OldSite$)
  UpdateProperties($T, T\_overlap$)
  $best\_score \leftarrow 0$
  $best\_candidate \leftarrow$ null
  for all $cand \in C$ **do**
    $cand\_overlap \leftarrow$ Overlap($cand, C$)
    UpdateProperties($cand, cand\_overlap$)
    $score \leftarrow$ VonSimiloScore($T, cand$)
    if $score > best\_score$ **then**
      $best\_score \leftarrow score$
      $best\_candidate \leftarrow cand$
    end if
  end for
  return $best\_candidate$
end function

function VonSimiloScore($T, cand$)
  $total \leftarrow 0$
  for all property lists (L_T, L_c) in Zip($T.properties, cand.properties$) **do**
    $m \leftarrow 0$
    for all $p_T \in L_T$ **do**
      for all $p_c \in L_c$ **do**
        $s \leftarrow$ Similarity(p_T, p_c)
        if $s > m$ **then**
          $m \leftarrow s$
        end if
      end for
    end for
    $total \leftarrow total + m$
  end for
  return $total$
end function

function UpdateProperties($e, OverlapSet$)
  for all $prop \in e.properties$ **do**
    $values \leftarrow [\,]$
    for all $o \in OverlapSet$ **do**
      append $o.properties[prop.name].value$ to $values$
    end for
    $prop.value \leftarrow values$
  end for
end function

function Overlap(e, S)
  $O \leftarrow [\,]$
  for all $cand \in S$ **do**
    if Overlaps($e, cand$) **then**
      append $cand$ to O
    end if
  end for
  return O
end function

function Overlaps($e, cand$)
  $r_T \leftarrow e.rect()$
  $r_C \leftarrow cand.rect()$
  $ratio \leftarrow$ IntersectArea(r_T, r_C) / UnionArea(r_T, r_C)
  $inside \leftarrow$ IsInside($cand.center(), r_T$)
  return $(inside \wedge ratio \geq 0.85)$
end function

The idea behind VON Similo (Algorithm 2) is that web elements often consist of multiple parts. The DOM orders its elements so that a node's children are inside

the area the parent occupies on the visually rendered website. They appear as one visual unit to the user and can be interacted with as one unit.

For example, a button might contain a button tag, an icon, and text. It does not matter which exact part the user clicks; the button will be triggered. When two elements have a considerable overlap, meaning they share a large part of their occupied area, they are likely to be part of the same visual unit. This unit is called visually overlapping nodes, or VON.

When a web element is moved to a different location on the website, the nodes in its visual overlap are often moved as well. The combination of elements moved together provides a more unique fingerprint than a single web element would. The situation is similar when a node is changed; its visual overlap might stay the same, making it easier to identify the modified element by the combination of elements in its visual overlap. For example, the button in our example moves to a different location on the website, but the icon and text stay the same. Alternatively, the text inside the button changes, but the button and icon stay the same.

The improved algorithm in VON Similo leverages this heuristic to identify elements more reliably. When a target element needs to be found on a new website version, the algorithm first identifies the visual overlap of the target element on the baseline version. Then, it iterates over all candidates on the updated version and calculates their respective visual overlaps. In the last step, a similarity score is computed between the target and each candidate, and the candidate with the highest score is returned as the target element in the new version.

More in detail, Algorithm 2 proceeds as follows. First, the visual overlap of the target element is computed on the original page (Line 2), and its properties are updated to include those of the overlapping elements (Line 3). Then, the algorithm iterates over all candidate elements (Lines 6–14). For each candidate, it computes the corresponding visual overlap (Line 7) and updates its properties accordingly (Line 8), before computing the similarity score (Line 9). The candidate with the highest score is retained (Lines 10–13). The score computation (Lines 17–32) differs from Similo in that properties are represented as lists. For each property, all pairwise similarities between target and candidate values are computed, and only the maximum similarity is retained (Lines 21–27). These values are then aggregated to produce the final score (Line 29). Finally, the procedures used to construct visual overlaps are defined in Lines 42–57.

To calculate the score, we need to define the visual overlap of a node. The VON Similo paper defines that two web elements E_1 and E_2 are visually overlapping if the following two conditions are met:

1. The areas R_1 and R_2, which their respective rectangles occupy on the screen in pixels, intersect to a certain degree. In other words if $\frac{R_1 \cap R_2}{R_1 \cup R_2}$ is above a certain threshold, which should be chosen in a way that balances:
   (a) accidentally grouping elements that do not belong together, when the threshold is chosen to lose, and there is no significant overlap, and
   (b) not recognizing two visually overlapping nodes as such by choosing the threshold too high.

   The authors propose a threshold of 0.85.
2. The center of E_2 is located inside of R_1. (The paper describes this differently - "The center of the web element W_1 (here E_1) is contained in the rectangle R_1" - but the underlying code shows that E_2 is compared with R_1).

In addition the LLM VON Similo paper introduces another metric to determine if E_1 and E_2 are visually overlapping based of the nodes visual text and XPath, which is used in addition to the algorithm used in VON Similo. We will called this approach textual overlap, and it is defined as:

1. The visible text of both elements is not `null` and their content is case-sensitive equal.
2. The absolute XPath of E_1 is a prefix of the absolute XPath of E_2, e.g, E_2 is a child of E_2 in the DOM.

After calculating all overlapping elements $E_1, ..., E_n$ for one element E, we replace the properties $E.a_1, ..., E.a_m$ of that element with lists of property values of the overlapping nodes, meaning the properties of E would look like this: $[E_1.a_1, ..., E_n.a_1], ..., [E_1.a_m, ..., E_n.a_m]$.

To compare two elements T and C, whose properties have been replaced with their respective lists, we modify the Similo calculation by choosing the pair from both property lists with the highest similarity. We then take the sum of those maximized values:

$$\text{VONSimilo}(T, C) = \sum_{i \in \#\text{properties}} \left(\max_{t \in T.a_i, c \in C.a_i} \text{similarity}(t, c) \right) \cdot c_i$$

3.3 LLM VON Similo

LLM VON Similo is the latest iteration of the Similo algorithm [25]. The algorithm first uses VON Similo to rank all elements on the website by their score. It then takes the top 10 visual overlaps, as the target element is most likely among them. The algorithm then utilizes GPT-4 [7] to find the target element among the pre-selected candidates. The algorithm converts the ten elements and the target element into JSON format. It then sends a request to the large language model with the target and the ten candidates, asking it to identify the target's match. The LLM responds with a single number, indicating the target's position in the list of candidate elements.

Since LLM VON Similo relies on proprietary OpenAI's APIs and uses an unspecified version of GPT-4 with unknown temperature and configuration settings, its results cannot be reliably reproduced in a controlled experimental environment. Therefore, we do not attempt a full replication of this method in this paper. Rather, in our replication, we only consider the benchmark used in LLM VON Similo and the VON Similo implementation and test the other algorithms against it.

3.4 Limitations and Threats to Validity

In this section we describe the limitations we identified about the the Similo algorithms, as well as the threats to the validity of the original studies that we aim to address in our replication work.

Threat T1 · Changed benchmarks. The benchmarks and underlying metrics vary across all three versions of Similo, making it challenging to accurately compare and evaluate each algorithm. We could identify three major inconsistencies between benchmarks, used metrics and evaluation strategies:

Changed data. The websites and elements used for the evaluation in the different benchmarks differs between the papers. The Similo and LLM VON Similo have approximately the same data (48 websites, ≈800 element pairs) with minor differences. The VON Similo paper only uses 36 websites and around 400 element pairs.

Changed metric. The Similo and LLM VON Similo benchmarks use a setup where the Similo algorithm is used to rank all elements on an updated version of a website by their similarity with the target, choosing the highest ranking one and validating whether it is the actual target. We used the same setup for all subsequent benchmarks. The VON Similo paper, on the other hand, calculates the similarity score between the target on the new and old version, declaring it a match when if the score reaches a certain threshold. We believe that this setup is not reflecting how Similo will perform in a web testing scenario. Additionally, it does not allow us to properly compare Similo with VON or LLM VON Similo as they have both never executed on the benchmark.

Changed similarity functions. The used similarity functions change slightly between the different iterations of Similo. For example, the original Similo algorithm uses Levenshtein similarity on the raw text, while the others apply it to lower cased strings.

Threat T2 · Coarse granularity of version snapshots. The benchmark used for all Similo papers utilizes website versions that were collected between 12 and 60 months apart. This scenario does not reflect actual web testing practices, where tests are run in a regular schedule, e.g., nightly or over the weekend. The changes made over a 12-60 month period are rarely made between two consecutive test runs of a test [9]. Hence, Similo can use the smaller update steps to repair itself by updating the saved value for a certain element. Utilizing the current benchmarks presents an unrealistic perspective on Similo's effectiveness in a more realistic testing time frame, thereby underestimating its true capabilities.

Threat T3 · Fixed set of properties and weights. The properties, similarity functions and weights used in the Similo algorithms were taken from the COLOR study [12] or given without empirical evidence that these values are suitable to be used in such an algorithm. Furthermore, they are reused for the VON Similo algorithm which works differently than the Similo algorithm. This reuse was implemented without assessing whether alternative properties, similarity functions, and weights might better suit the VON Similo's distinct computational approach, which emphasizes visual overlap.

Threat T4 · Limited accuracy with VON Similo. Algorithms that use visual overlap can identify only the group of elements that overlap visually, not the specific target within that group. For certain test actions, such as clicking a button or link, this does not represent an issue because the website interprets any click within the overlapping area as a click on the element itself. However, for other actions like entering text into an input field, a text area, or verifying specific properties of a particular element, merely interacting with any element in the overlapping area is insufficient. Instead, identification of the exact element is necessary.

The underlying problem is that given T and C' as well as elements $E^T_{1..n}$ and $E^{C'}_{1..m}$ which form their respective visual overlaps O^T and $O^{C'}$, then this visual overlap is the same for all elements in the overlap, e.g. $\forall C \in O^{C'} : \texttt{overlap}(C) \equiv O^{C'}$. When Similo compares T, with every $E^{C'}_{1..m}$, it will always compare their visual overlaps: O^T and $O^{C'}$ and calculate the same score each time. Because the algorithm chooses the element with the highest score, it selects a random element from the visual overlap $O^{C'}$.

3.5 Implementation

To be able to evaluate the Similo algorithm in a practical setting, we implemented a library for Java, compatible with the Selenium WebDriver framework. The library wraps standard Selenium locators and automatically uses the Similo algorithm to identify the correct web element when the original locator fails. Upon first usage or locator changes, the library captures relevant element properties and saves them in an SQL database to do accurate matching in subsequent test executions. The wrapper supports all Selenium's locator strategies (e.g., XPath, CSS selectors, ID), minimizing integration effort with existing test suites.

The implementation features built-in self-repair capabilities by locating the element whose locator is broken and using it to continue test execution. It also intelligently updates locators within the database when elements change by determining the most stable of the basic locators XPath, ID and ID-XPath and using it for future localizations. Doing so, it enhances test resilience without modifying test code directly but remembering the connection between the locator and its current state of properties. Moreover, the library monitors Similo scores and issues configurable warnings when low score matches occur, guiding developers toward potentially broken locators. The library is available on GitHub and has comprehensive documentation [29].

4 Empirical Study

4.1 Research Questions

We consider the following research questions:
RQ$_0$ (replication): Can we replicate the experimental results yielded by state-of-the-art tools targeting robust locator generation?
RQ$_1$ (comparison): How do Similo, VON Similo, and LLM VON Similo compare with each other on the same benchmark?
RQ$_2$ (improvements): How do effectiveness vary when considering different metrics in Similo and VON Similo?
RQ$_3$ (hybrid): How do effectiveness vary when combining Similo and VON Similo?

In the first research question (RQ$_0$) we aim to confirm the reliability of existing robust locator generation approaches Similo and VON Similo by reproducing their experimental results against their original data sets. The second research questions (RQ$_1$) addresses **T1** by performing a comparison by executing the selected algorithms on four benchmarks, three taken from the original papers, and

a new one substantially extended in this work (hence addressing **T2**). The third research question (RQ_2) evaluates a large set of configurations of the original algorithms, varying the metrics being used to optimize the accuracy of the algorithm on different metrics. This question aims to address the problems discussed in **T3**. The last research question (RQ_3) evaluates HybridSimilo, a novel hybrid approach in which we combine Similo and VON Similo. This question aims to address the problems discussed in **T4**.

4.2 Benchmarks

To mitigate the problems associated with **T1**, in this paper we benchmark all algorithms on all available benchmark available in the original Similo, VON Similo, and LLM VON Similo papers. Particularly, all benchmarks use the Web Archive [37] to collect the element pairs and their property values. We also mitigate **T2** by providing an extended benchmark constructed using the websites as the original benchmarks and the same selection criteria for web elements, but sampling a higher number of web elements and versions over time. All the locators used in the original benchmarks were taken from the corresponding replications packages. The used websites were reloaded from the same WayBack Archive links.

4.2.1 Similo's Benchmark

The benchmark used to evaluate the original Similo algorithm consists of 809 web element pairs T and C', from 48 websites, each having 12 to 60 months between versions.

4.2.2 VON Similo's Benchmark

For the VON Similo benchmark 442 web elements from 33 websites were used, each having 12 to 60 months between version, similar to the Similo benchmark. Additionally all the elements from the visual overlaps of the targets were added, resulting in 1,163 element pairs. Finally for every matching element pair, one randomly selected non matching pair was added. The non matching pairs were selected to measure the false positive and false negative rate of the algorithm. The 442 base web elements of the VON Similo benchmark are a subset of the 809 web elements of the Similo benchmark. The paper does not provide any information why a smaller benchmark was used for VON Similo.

4.2.3 LLM VON Similo's Benchmark

The LLM VON Similo benchmark utilizes the same website versions as Similo and shares 804 element pairs. The other five elements which were part of the original benchmark could not be located due to changed rendering when reloading the websites from the WayBack Archive.

4.2.4 Extended Benchmark

We collected web elements from the homepages of 30 popular web applications, considering the 48 websites used in original papers and combining it additional websites from a website ranking from 2023 [35]. We had to discard 18 websites that contained broken snapshots or did not render properly. Five, only partially broken websites were used for later cross-validation of our training approach (further details are available in our replication package).

To better reflect to short time spans between test executions and to improve comparability, we chose a fixed time span between two version of four months and selected 16 version from September 2018 to September 2023 for each web application. We then tracked elements across those versions, resulting in a total of 933 inital elements from these websites and 10,376 element pairs of the entire time span. This process took three months of manual work for mapping each element across all versions. We also classified the element pairs in categories, determined if any of the basic unqiue locators (ID, XPath, or ID-XPath) changed or if it was still locatable using the one of these. This information is used to later benchmark the performance of the algorithms on locator pairs with broken unique locators. The total distribution of elements across the websites is shown in the appendix in Table 4.

4.3 Algorithms Extensions

To mitigate the problems associated with **T3** and **T4**, we devised three extensions of the original Similo and VON Similo algorithms, introducing a hybrid method that leverages both.

4.3.1 Similo++

The first improvement to Similo is called Similo++. In brief, we improve the properties compared by the algorithm, the comparison algorithms used to compute the similarity between two properties and the weights which are multiplied with the similarity.

Compared Properties. Along with the properties used in the original Similo algorithm, we analyzed the frequency (how many element pairs have that property) and stability (in how many cases has the property the same value on the updated version). We found two additional suitable attributes: `type` (frequency of 6-7% and stability of 95-96%) and `aria-label` (frequency of 10-12% and stability of 81-84%) across five random subsets of the training data as well as the cross validation set. Additional we compare all attributes for that element as a key-value map.

Similarity Functions. The original paper utilizes simple (lower case) equality, Levensthein distance and word comparison to compare string properties, Euclidian distance for integer fields like area and shape as well as 2D-Distance for the coordinates. We extended these comparison algorithms by consider additional distance metrics. For string based properties, we considered the following additional comparison algorithms:

Jaccard. The Jaccard distance is originally a measure to compare two sets and is defined as the size of the intersection divided by the size of the union of the

input sets A and B, $\frac{|A \cap B|}{|A \cup B|}$. To use the Jaccard distance to compare strings, we first split the strings into sets of characters and then use the Jaccard distance to compare the sets. The Jaccard distance between "kitten" and "sitting" is $\frac{3}{7}$, because the intersection of the sets is {"t", "i", "n"} and the union is {"k", "i", "t", "e", "n", "s"}.

Jaro Winkler. The Jaro-Winkler distance is a string metric measuring the edit distance between two sequences. The Jaro-Winkler distance is given by a modification of the Jaro distance formula, where more weight is given to strings that match from the beginning. The Jaro distance is given by $d = 1 - \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m})$, where m is the number of matching characters and t is half the number of transpositions. A transposition is a pair of matching characters in the wrong order in one of the strings. The Jaro-Winkler similarity is then calculated as Jaro similarity $+ l \times p \times (1 -$ Jaro similarity), where l is the length of the common prefix at the start of the string up to a maximum of four characters, and p is a constant scaling factor, often 0.1. For example, the Jaro-Winkler similarity between "kitten" and "sitting" is approximately 0.74, assuming the Jaro distance is 0.77, and there is no common prefix.

Set similarity. The set similarity is similar to the Jaccard similarity. The strings are split into sets of strings at spaces and newlines. The final result is the Jaccard distance between the two sets. For example, the similarity between "Sign up" and "Sign in" would result in $\frac{1}{3}$, because the intersection is {"Sign"} and the union is {"Sign", "up", "in"}. Capitalization is ignored.

To compare properties which consist of key value pairs, i.e. the elements attributes, we considered the following two algorithms:

Intersect Value Compare. It is calculated as

$$\frac{|\{(k,v)|(k,v) \in A \cap B\}|}{\max(|A|,|B|)}$$

where $A.k$ is the value of the key k in the map A.

Intersect Key Compare. It is calculated as

$$\frac{|\{k|k \in A \cap B\}|}{|\{k|k \in A \cup B\}|}$$

To compare the distance between two elements we introduced two new similarity algorithms:

Manhattan Distance. The distance between two points in a grid based on a strictly horizontal and/or vertical path. The Manhattan distance between the points (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$. The result is normalized to $[0, 1]$ by dividing it by a predefined maximum distance.

Exponential decay. uses the Euclidean distance but adds exponential decay, with differing λ values calculated as $e^{-\lambda d}$, where d is the Euclidean distance between the points. The benefit is that the function is already normalized to $[0, 1]$ and approaches 0, eliminating the need to define a maximum distance. We used different values for λ, evaluating three exponential decay similarity functions, namely $\lambda = 0.001$ as a small decay, $\lambda = 0.005$ as a medium, and $\lambda = 0.01$ as a large decay.

Finally to compare the shape of the two elements we used:

Area. The area of an element is the product of the width and height of the minimal rectangle containing the visual element.

Perimeter. The perimeter of an element is the sum of the length of all sides of the minimal rectangle containing the visual element.

Aspect Ratio. The aspect ratio of an element is the ratio of the width to the height of the minimal rectangle containing the visual element. To compare the respective values of two elements, we divide the smaller value by the larger value. For example, comparing an element with an area of 100px and an element with an area of 200px would result in $\frac{1}{2}$ because the smaller element has half the area of the larger element.

Improved Weights. We use a genetic algorithm to optimize the weights of the Similo algorithm and evalute whether a global combinations of weights can be found. We applied the genetic algorithm as follows. We begin by fine-tuning the similarity functions for each property, starting from an initial baseline of weights and functions. For each property, we evaluate every possible similarity function, choosing the most effective one. This method is systematically applied to each property in sequence, optimizing them one at a time. The selection of property similarity functions might change based on the order in which the properties are optimized. To ensure a robust outcome, we randomly select a property for comparison and conduct multiple rounds of optimization. In cases where several similarity functions perform well for a property, we employ a brute force approach to determine the best combination from this narrowed selection of functions. We define a similarity function for each attribute and then apply genetic optimization to determine the optimal set of weights. We use fixed step values with 0.05 step size in $[0, 3]$ as possible values for weights. Essentially, the algorithm assigns a weight of zero to any attribute that does not enhance overall fitness, thereby excluding it from consideration in the algorithm. The specific fitness function that evaluates a given weight combination vary based on the optimization goal. More details are provided in Section 4.4.

4.3.2 VON Similo++

VON Similo++ is an improved version of VON Similo, optimized with the same process as Similo++ but with a different objective function: instead of trying to locate the concrete target, an element will also be allowed as a match if it is in the visual or textual overlap or the target is among the top ten highest ranked elements.

4.3.3 HybridSimilo

The HybridSimilo approach aims to overcome the limitations of VON Similo mentioned in **T4**. As discussed, approaches using visual overlap can only identify all elements in the visual overlap with the same accuracy, but not the exact element described by the locator. This can lead to problems in certain testing scenarios, as the element selected in the visual overlap might not be the exact target. On the other hand, Similo can identify the one single concrete element with high accuracy. A preliminary benchmark showed that the basic VON Similo algorithm could directly identify the target element in 85.5% of cases and rank it among the top five in 95.5% of cases. At the same time, Similo was able to identify the target element in 88% of cases directly and in 94% among the top five. While the

basic VON Similo algorithm is superior in selecting all the elements in the visual overlap, it is inferior in selecting the exact target element.

The idea is to overcome the limitations of VON Similo by combining both algorithms, leveraging the strengths of both. We pre-select the elements in the visual overlap using VON Similo and then identify the concrete target among them using Similo. In theory, the visual overlap identified by VON Similo should contain elements which differ in their tags, attributes and properties, as the contribute to the formation of a visually cohesive unit. Similo should be able to correctly select the concrete target from the this pre-selection made by VON Similo. As the algorithm combines Similo and VON Similo, we named it HybridSimilo.

4.4 Metrics

Across all two replicated papers, the authors used different evaluation metrics. The metrics proposed in the existing papers are:

Metric 1 · Similo. For each of the pairs, the algorithm is tasked to find the the C' among all candidates. If the selected candidate is equal to C' or a direct parent or child in the DOM structure of C' the selection is considered a match.

Metric 2 · VON Similo. For the evaluation the authors compared all pairs of original and updated elements by calculating the normalized Similo and VON Similo score between the pair and classifying the pair a match if the score exceeded a certain threshold. They found that for VON Similo a threshold of 0.4 was optimal and for Similo a threshold of 0.28.

In our replication, we consider three additional metrics.

Metric 3 · . Visual or Textual Overlap. Similar to metric 1, the algorithm is tasked to find the correct element among all candidates on an updated version of a websites given the target. A localization is deemed correct if the visual overlap or textual overlap of the located element contains the concrete target. This metrics deems significantly more localization's as correct compared to the more restrictive metric used in Similo. This metric was also used in the successor LLM VON Similo.

Metric 4 · Exact Match. This new metric determines a located candidate a match if the located element without overlapping elements is the exact target. This is the most restrictive metric but also the most accurate one. A high score indicates that the algorithm is able to correctly identify the target element and in a testing scenario all possible use cases of a locator can be handled (i.e. clicking on an element, entering text, comparing properties).

Metric 5 · Locator Changed - Exact Match. This metric focuses on an algorithms performance specifically for elements where traditional unique locators (ID, XPath, or ID-XPath) have failed due to website updates. It is similar to Metric 4, which considers all element pairs, but Metric 5 focuses exclusively on the subset of elements that represent the cases where a real test suite would fail and manual locator repair would be needed. By focusing on these cases, we can evaluate the effectiveness of different algorithms specifically in situations where conventional locator strategies fail between updates to the website. This metric provides a more focused assessment of the algorithm's capability to address the main challenge in web test maintenance, correctly recovering from broken locators without manual intervention.

Metric 6 · Fitness. In the extended benchmark, we collected detailed information about the variations between different versions, specifically focusing on the elements and their locators. We categorized the element changes into three types, namely *No change*, *Minor change*, and *Major change*. *No change* refers to elements that retain identical attributes across versions. *Minor change* includes elements that maintain the same tag, text, and attributes, with their locations shifting by no more than 10 pixels in any direction and dimensions changing by no more than 5 pixels, including modifications in the CSS style. We classify as *Major change* all other evolution patterns.

Additionally, the elements were categorized based on their locators into three groups, namely *All locators work*, *Absolute XPath does not work*, *No locators work*. We then assigned a localization score to each element pair based on these categories (we report the actual scores in our replication package). Higher scores were given for more significant changes. The overall fitness score for each pair is the aggregate of their localization scores, rewarding the full score if the exact element was identified and one-quarter of the score if there was only partial overlap.

In this replication study, we evaluate the two original algorithms (i.e., Similo and VON Similo), along with the our new extensions, using all evaluation metrics.

4.5 Procedure

Concerning RQ_0, we executed Similo on the original Similo benchmark, Similo and VON Similo on the original VON Similo benchmark.

Concerning RQ_1, we executed the different Similo and VON Similo from each paper on the Similo and VON Similo benchmarks as well as our extended benchmark. For all combinations we captured all metrics M1-M6, if feasible.

Concerning RQ_2, we employed the optimization process described in Section 4.3.1 to the Similo and VON Similo algorithm on different benchmarks and using different fitness metrics. Specifically we optimized Similo on the LLM VON Similo benchmark, as it is the most recent one, for metric M3 and M4 and VON Similo for M3, to provide comparability with the other algorithms. On the extended benchmark we optimized Similo for M6.

Concerning RQ_3, we utilized the optimized algorithms from RQ_2, specifically VON Similo optimized on the Similo benchmark to improve M3, as well as Similo optimized on the Similo benchmark to improve M4. We then combined these to first select a set of candidates with VON Similo and then determine the exact match among those candidates with Similo.

Overall, our experiment includes nine algorithm configurations under test, four original ones by selecting Similo and VON Similo from the original papers as well as seven new ones. As our evaluation set comprises 10,376 element pairs overall, we ran 280,233 localization attempts in our replication and extended study.

5 Results

Table 2 shows the results for $RQ_{0\text{-}1\text{-}3}$. The columns show the results for different metrics with the number of localization complying with the metric first and the percentage of the total number of localizations in brackets. The metric in

parenthesis indicates the benchmark the algorithm was optimized on as well as the metrics being optimized by the genetic algorithm. I.e. (LLM, M4) means that the algorithm was optimized on the LLM VON Similo benchmark and the metric M4 was used as a fitness function. HybridSimilo is not applicable to M2, because the process of pre-selection does not work with the study setup, where a concrete element pair is given and the score needs be calculated. The columns are each for a specific variation of Similo. For the Similo algorithms taken from the original papers (Similo and VON Similo) are followed by the specific paper they are taken from in brackets. For the extended algorithms (Similo++, VON Similo++, HybridSimilo) the content of the brackets indicates on which benchmark the algorithm was optimized and what metric was used as a fitness function. Results are reported separately for each considered benchmark. For metrics M2 and M6 the underlying numbers, i.e., the number of elements correctly classified by the threshold and the exact fitness have no informative value, why we only report the percentage.

5.1 Replication (RQ_0)

For Similo, the replicated algorithm and benchmark found 88.99% elements, where the original paper was able to locate 88.64% of them. For VON Similo used in the LLM VON Similo paper, our replication found 91.65% of elements, where the original paper found 91.29% of elements. These minor differences were expected, as we reloaded the benchmarks from the Web Archive and differences in browser versions and window size can alter coordinates, shapes and areas as well as neighboring text used in the algorithm. When replicating Similo and VON Similo algorithms on the VON Similo benchmark, we also found challenges associated with the use of random non-fitting element pairs in the original benchmark, as the specific elements can change the results.

RQ_0 (replication): *Our reproduced results closely match the original outcomes for both Similo and VON Similo, indicating that our replication was successful.*

5.2 Comparison (RQ_1)

Our results show that our assumption that VON Similo performs worse than Similo in locating a concrete element and not just its visual overlap is correct. In our experiments, VON Similo under performs Similo on all benchmark, except the VON Similo benchmark, on the metric M3. This reinforces our initial hypothesis that Similo excels not only in identifying exact elements but also in recognizing their visual overlaps. VON Similo surpasses Similo in scenarios involving broader overlap criteria, such as M3 (visual or textual overlaps). This suggests that while Similo tends to either rank the correct targets very high, VON Similo consistently ranks them high, albeit not in the top position.

RQ_1 (comparison): *Similo is more effective than VON Similo at retrieving the exact target element, while VON Similo performs better when broader visual or textual overlap is sufficient.*

Table 2: $RQ_{0\text{-}1\text{-}3}$: Results for all algorithms across all benchmarks and evaluation metrics (best results are highlighted in bold).

	M1 (Similo)	M2 (VON)	M3 (LLM)	M4 (Exact)	M5 (LC Exact)	M6 (Fitness)
Benchmark used for the Similo Paper (809 Element Pairs, 510 with broken locators)						
Similo (Similo)	720 (88.9%)	83.7%	720 (88.9%)	701 (86.6%)	406 (79.6%)	83.1%
Similo (VON)	727 (89.8%)	85.6%	725 (89.6%)	705 (87.1%)	410 (80.4%)	83.8%
VON Similo (VON)	722 (89.2%)	92.8%	722 (89.2%)	627 (77.5%)	353 (69.2%)	76.5%
Similo++ (LLM, M4)	758 (93.7%)	75.6%	754 (93.2%)	734 (90.7%)	436 (85.5%)	88.4%
Similo++ (LLM, M3)	760 (93.9%)	83.6%	758 (93.7%)	715 (88.4%)	417 (81.7%)	85.5%
VON Similo++ (LLM, M3)	728 (89.9%)	89.1%	**765 (94.6%)**	540 (66.7%)	299 (58.6%)	65.7%
HybridSimilo (LLM, M4)	**766 (94.7%)**	N/A	764 (94.4%)	**742 (91.7%)**	**443 (86.8%)**	**89.7%**
Similo++ (Ext, M6)	743 (91.8%)	73.2%	737 (91.1%)	713 (88.1%)	418 (81.9%)	85.4%
HybridSimilo (Ext, M6)	749 (92.6%)	N/A	745 (92.1%)	719 (88.8%)	424 (83.1%)	86.4%
Benchmark used for the VON Similo Paper (441 Element Pairs, 272 with broken locators)						
Similo (Similo)	395 (89.6%)	83.4%	395 (89.6%)	384 (87.1%)	217 (79.8%)	82.4%
Similo (VON)	399 (90.5%)	85.2%	397 (90.0%)	385 (87.3%)	218 (80.1%)	82.8%
VON Similo (VON)	399 (90.5%)	93.6%	397 (90.0%)	323 (73.2%)	173 (63.6%)	71.9%
Similo++ (LLM, M4)	415 (94.1%)	76.1%	410 (92.9%)	398 (90.2%)	229 (84.2%)	86.1%
Similo++ (LLM, M3)	416 (94.3%)	84.7%	412 (93.4%)	381 (86.4%)	212 (77.9%)	81.4%
VON Similo++ (LLM, M3)	403 (91.4%)	92.3%	**415 (94.1%)**	254 (57.6%)	126 (46.3%)	54.5%
HybridSimilo (LLM, M4)	**417 (94.5%)**	N/A	414 (93.8%)	**402 (91.1%)**	**233 (85.6%)**	**88.1%**
Similo++ (Ext, M6)	406 (92.1%)	74.1%	399 (90.5%)	385 (87.3%)	219 (80.5%)	83.3%
HybridSimilo (Ext, M6)	407 (92.3%)	N/A	402 (91.1%)	386 (87.5%)	220 (80.8%)	83.7%
Benchmark used for the LLM VON Similo Paper (803 Element Pairs, 500 with broken locators)						
Similo (Similo)	722 (89.9%)	84.2%	722 (89.9%)	703 (87.5%)	404 (80.8%)	84.2%
Similo (VON)	729 (90.8%)	86.2%	727 (90.5%)	707 (88.0%)	408 (81.6%)	84.9%
VON Similo (VON)	724 (90.1%)	92.9%	724 (90.1%)	629 (78.3%)	351 (70.2%)	77.5%
Similo++ (LLM, M4)	760 (94.6%)	75.7%	757 (94.2%)	737 (91.8%)	435 (87.00%)	89.7%
Similo++ (LLM, M3)	762 (94.9%)	84.1%	761 (94.7%)	718 (89.4%)	416 (83.2%)	86.8%
VON Similo++ (LLM, M3)	731 (91.0%)	89.1%	**768 (95.6%)**	544 (67.7%)	298 (59.6%)	66.8%
HybridSimilo (LLM, M4)	**768 (95.6%)**	N/A	767 (95.5%)	**745 (92.8%)**	**442 (88.4%)**	**91.1%**
Similo++ (Ext, M6)	745 (92.8%)	73.6%	740 (92.1%)	716 (89.1%)	417 (83.4%)	86.7%
HybridSimilo (Ext, M6)	751 (93.5%)	N/A	748 (93.1%)	722 (89.9%)	423 (84.6%)	87.7%
Extended Benchmark (10376 Element Pairs, 2012 with broken locators)						
Similo (Similo)	10276 (99.0%)	89.1%	10275 (99.0%)	10274 (99.0%)	1926 (95.8%)	97.6%
Similo (VON)	10286 (99.1%)	**91.4%**	10285 (99.1%)	10282 (99.1%)	1934 (96.1%)	97.8%
VON Similo (VON)	10168 (97.9%)	90.3%	10270 (98.9%)	9448 (91.0%)	1689 (83.9%)	90.6%
Similo++ (LLM, M4)	10322 (99.5%)	79.1%	10324 (99.5%)	10321 (99.4%)	1957 (97.3%)	98.6%
Similo++ (LLM, M3)	10328 (99.5%)	85.1%	10322 (99.5%)	10320 (99.4%)	1958 (97.4%)	98.6%
VON Similo++ (LLM, M3)	9779 (94.2%)	90.8%	10308 (99.3%)	8,000 (77.1%)	1415 (70.4%)	78.1%
HybridSimilo (LLM, M4)	10311 (99.4%)	N/A	10307 (99.3%)	10304 (99.3%)	1940 (96.4%)	98.1%
Similo++ (Ext, M6)	**10356 (99.8%)**	79.7%	**10356 (99.8%)**	**10352 (99.7%)**	**1988 (98.8%)**	**99.4%**
HybridSimilo (Ext, M6)	**10356 (99.8%)**	N/A	**10356 (99.8%)**	**10352 (99.7%)**	**1988 (98.8%)**	**99.4%**

5.3 Improvements (RQ_2)

Table 3 shows the concrete values we found for each of the property and optimized algorithm. The header shows the algorithm we optimized, the benchmark it was done on and the metric we used as the optimization objective.

We found that different sets of properties, similarity functions and weights can significantly improve the capabilities of Similo and VON Similo. On the LLM VON Similo benchmark, the optimization of Similo improved the M4 (exact match) metric from 87.54% to 91.78% and the M3 (overlap visual and textual) from 91.65% to 95.64%. While the original algorithms scored very high, in our study we show that there is room for improvement by optimizing the chosen properties, weights and similarity functions. Because this optimization used a very broad set of websites,

Table 3: RQ_2: Optimized weights and similarity functions for Similo and VON Similo.

	Similo (Ext M6)		VON Similo (Sim. M3)		Similo (Sim. M4)		Similo (Sim. M3)	
Property	Wei.	Sim.	Wei.	Sim.	Wei.	Sim.	Wei.	Sim.
Tag	0.80	Jaccard	1.25	Levenshtein	2.35	Jaro Winkler	0.80	Levenshtein
Class	-	-	0.65	Jaro Winkler	1.00	String Set	1.1	Levenshtein
Name	2.85	Levenshtein	1.80	Equality	2.90	Equality	1.70	Equality
ID	0.50	Levenshtein	2.50	Jaccard	2.70	Levenshtein	2.85	Levenshtein
HRef	0.95	Equality	0.80	Levenshtein	0.30	Levenshtein	2.85	Levenshtein
Alt	1.85	Equality	0.10	Levenshtein	1.95	Levenshtein	0.60	Levenshtein
Type	2.75	Equality	2.85	Equality	1.10	Equality	2.45	Equality
Aria-Label	0.90	Jaccard	2.35	Levenshtein	2.95	Equality	1.40	Equality
Abs. XPath	0.10	Jaccard	1.05	Levenshtein	0.50	Equality	0.05	Equality
ID-XPath	0.45	Levenshtein	0.75	Equality	0.50	Levenshtein	1.25	Levenshtein
Is Button	-	-	2.85	Equality	-	-	0.10	Equality
Location	1.20	Medium Decay	2.00	Small Decay	2.00	Manhattan	2.15	Linear
Dimension	0.35	Area	0.95	Area	1.30	Area	1.85	Area
Visible Text	2.80	Levenshtein	2.50	Levenshtein	2.95	Levenshtein	2.55	Levenshtein
Neighbor Text	1.45	String Set	2.30	Levenshtein	1.00	Levenshtein	1.70	String Set
Attributes	1.80	Intersect Value	1.00	Intersect Value	2.20	Intersect Value	2.50	Intersect Value

with different specifics due to different web frameworks and different web design, we assume that the optimization would be even more effective for a smaller, more specific set of websites or for a specific web framework.

Some properties like class or is button are often ranked low or excluded entirely from the algorithm. Other properties such as the name, type, aria-label, location, visible text, neighbor text and attributes have high weights no matter the metric, indicating that they are important properties for Similo. It is important to note that the optimization process is random and might return different local optima, depending on the initial values. This explains outliers like the sudden high weight of "is button" for VON Similo (Sim. M3).

RQ_2 (improvements): *Optimizing the property set, similarity functions, and weights yields clear improvements for both Similo and VON Similo, showing that their default configurations are not optimal. Key attributes (e.g., name, type, aria-label, text, location) consistently receive high importance, while others contribute little. Gains are likely even larger when optimizing for specific websites or frameworks.*

5.4 Hybrid (RQ_3)

Combining Similo and VON Similo can improve the accuracy of the locator relocalization, but only in specific configurations. On the extended benchmark, Similo performs better than VON Similo in identifying the visual overlaps. Therefore, we utilize VON Similo to find the the top ten highest ranking elements, as it is the only task where VON Similo outperforms Similo. Nonetheless, the resulting HybridSimilo algorithm, which utilizes VON Similo++ to pre-select ten candidates and Similo++ to locate the concrete target, HybridSimilo performs similar to Similo++ for all metrics.

On the original benchmarks, specifically the LLM VON Similo benchmark, VON Similo++ proves to be the best algorithm at selecting the visual or textual overlap (M3). Based on the pre-selected overlap by VON Similo++, we then used Similo++ to find the concrete element. This specific HybridSimilo version outperforms all other algorithms on the M1 and M4-M6 metric on all original benchmarks. Particularly, it is able to locate 95.51% of elements on the LLM VON Similo benchmark under the M3 metric, slightly outperforming LLM VON Similo (95.0%) [25]. These results suggest that VON Similo is effective in identifying an initial selection when there is a significant difference between versions. This selection can then be refined by Similo. However, VON Similo does not provide additional advantages in case of minor updates between versions.

To ensure that we did not overfit on the benchmark data, we conducted temporal cross-validation on the extended benchmark, using the last n snapshots over all sites as test data and training on all earlier snapshots in five folds. This setup best reflects the common use case of the algorithm, where it is deployed in a functioning test suite and later tasked with locating changed elements on the same website. This approach ensures that our results translate to common testing scenarios. The results show that the performance of the optimized versions on the extended benchmark differs by only 0.1% on all metrics when using cross-validation. Similo++ and HybridSimilo again performing similarly.

RQ_3 (hybrid): *Hybridizing Similo and VON Similo offers benefits only in specific cases. VON Similo is useful for generating an initial shortlist when versions differ substantially, and Similo can then refine this selection to identify the exact element. This hybrid approach improves accuracy on benchmarks with large visual or structural changes, but provides little advantage when updates are minor. Temporal cross-validation confirms that these effects generalize, with hybrid and optimized Similo performing similarly.*

5.5 Threats to Validity

5.5.1 Internal Validity

We compared all variants of the replicated and extended algorithms under identical experimental settings and on the same evaluation sets, which was not the case in the original studies. The main threat to internal validity concerns our implementation of the original algorithms and evaluation scripts, which we tested thoroughly. Our replication of the original results confirms the correctness of our replication efforts.

5.5.2 External Validity

The limited number of websites in our evaluation poses a threat in terms of generalizability of our results to other web apps. We assume that the datasets comprising the most popular websites provides a realistic representation of how websites change over time, as seen in a continuous integration environment. However, different web frameworks and libraries that are not represented in our dataset could

yield different results, and systematically evaluating them is left for future research. Furthermore, only the static front pages of websites were used for the dataset. Front pages typically consist of links, headers, images, and menu items and may not represent the diversity of elements found in other parts of a web application. This selection bias could affect the algorithm's effectiveness on pages different than front pages, where elements such as selects, tables, and table items appear more frequently. The final algorithm, with its optimized weights and similarity functions, is tailored to this dataset. Despite using cross-validation to prevent overfitting, there is a risk that the results are overly optimized for the dataset, potentially affecting the algorithm's performance on different websites. The data was scraped from the Wayback Machine, which may not always capture a website's complete or accurate representation. Additionally, the snapshots are scraped across different browsers in different countries and further pre-processed before saving and rendering them, introducing further inaccuracies into the dataset.

All these factors could lead the optimized algorithm to perform differently in sanitized testing environments.

6 Discussion

Discussing main differences with existing literature. Our results confirm that our Similo and VON Similo replications closely match the original studies and Similo consistently outperforms VON Similo on localizing a concrete element, while VON Similo is only superior on broader overlap metrics (M2/M3). When optimizing properties, similarity functions, and weights yields small but meaningful gains at high baselines (e.g., Similo++ and VON Similo++ improve exact-match accuracy by up to 5.6 percentage points and achieve up to 71% relative improvement on broken-locator cases). Moreover, the hybrid variant using VON Similo++ for candidate pre-selection and Similo++ for final ranking delivers the best overall performance on the original benchmarks and remains competitive on the extended dataset. This indicates that VON Similo is most useful as a overlap pre-filter under larger changes.

Column M2 (i.e. the metric used for the VON Similo algorithm) differs significantly from the other metrics, as it is the only metric on which VON Similo outperforms all other algorithms. This difference is primarily due to variations in the study setup compared to those used for Similo and LLM VON Similo. The VON Similo reported than 94.1% of element pairs were accurately classified as either matching or non-matching. Given that in case of a localization in a test case the target T will be compared with all C, we would require $|C| - 1$ correct classifications as non-matching (sensitivity being 0.97) and one as matching (recall being 0.922). With approximately 800 elements in C to compare with T, the probability of correctly identifying the target in the new version is calculated as $0.97^{799} \cdot 0.92$, which approximates to 0.

Our results show that VON Similo is not substantially better than Similo, even for identifying the visual overlap. Based on our efforts in hybridizing Similo and VON Similo, we suggest that these algorithms should always be used jointly, as the concrete selection with Similo is expected to improve the locator detection accuracy.

Consequences for use in practical web testing. The successfully replicated results show that the proposed algorithms are effective at identifying web elements, even when their standard locators (ID, XPath, ID-XPath) are not working. The original Similo algorithm is able to correclty identify 95.8% of elements whose original locators are not working anymore. This means that only 4 out of 100 locator breakages need to be manually repaired by a developer, significantly reducing the cost of maintaining a web application. The optimized version of Similo even reduces this number to 1 out of 100. For example, other prominent locator strategies like Tag + Text only achieve a success rate of around 82%. With the help of the implemented library, developers can directly incorporate the algorithms into their testing process to save time and effort.

Empirical maintenance studies put concrete numbers on the cost of manual locator repair. In an industrial case study involving four real-world Selenium suites, Leotta et al. report that repairing a single release after locator breakage requires 0.60 h when ID locators are used and 3.05 h when XPath locators are used [13]. At the current U.S. market rate for a test-automation engineer (average $90 k p.a. [11], equivalent to $43.5 h^{-1}), this corresponds to $26 and $133 per fix. Assuming a medium-sized organization with 500 test suites × 10 locators (5,000 locators overall) releasing weekly, and observing that 26% of XPath locators break (1,300 failures) and in 19% neither ID nor XPath work (950 failures), maintaining pure, XPath suites would cost approximately $8.6 million annually, compared to $1.2 million for ID-based suites. These findings echo Accenture's independently reported $50–$120 million annual spend on GUI-test maintenance [8].

With the original Similo algorithm—automatically recovering 96% of those 950-1300 weekly dual-break failures—manual repairs drop to just 38-52 per release ($988-$6,916). The optimized Similo variant (99% recovery) further reduces these to around 10-13 repairs ($260-$1,729) per release. Over a 50-release year, this cuts annual maintenance from $1.2-$8.6 million (no automated healing) to about $49,400-$345,800 with Similo or only $13,000-$86,450 with optimized Similo, a 96–99 % reduction in spend. Even marginal gains in locator-healing accuracy thus translate into five-figures annual savings in good maintained test suites or six-figure annual savings in those using fragile locators. This underlines the clear industrial value of further algorithmic improvements.

Impact of long and short inter-version time intervals on performance and benchmark. At first glance, it may appear that improved methods offer negligible benefits for benchmarks with shorter inter-version intervals since baseline algorithms already perform strongly. However, considering improvements relative to the maximum achievable performance reveals their true significance. For instance, in long-term intervals, performance improved from 86.6% to 91.7%, representing 38% of the possible remaining margin towards 100%. For short-term intervals, the baseline already achieved 99.0%, and improved to 99.7%, again representing a substantial 70% of remaining possible improvement. Furthermore, focusing specifically on elements with changed locators, performance improved from 95.8% to 98.8%, covering an even larger relative improvement of 71%. These results underline that even minor absolute improvements at high baseline levels can translate into significant practical gains, particularly when addressing more challenging locator updates. Additionally, we can see that while the Similo algorithms perform better on short-term changes, repairing locators with 98.8% accuracy, their performance on long-term changes of 1–5 years is significantly lower at 88.4%, though

still better than baseline approaches. This underlines the need for frequent updates to the attribute set of a locator, as done in our implementation.

Error analysis. Despite optimization efforts, certain web elements remain consistently misidentified. Common reasons for such errors include:

- Substantial changes in attributes between versions (e.g., altered text or tags). This can for example happen when a "Log in" button changes from a `<button>` tag to a styled `<span>`, significantly reducing similarity scores.
- Similarity in visual placement outweighing textual cues. For example, a new "Sign up" button positioned exactly like the old "Log in" button might cause the algorithm to prioritize visual similarities over differing text.
- Inability to interpret purely visual elements like icons. When visually similar icons without textual differences swap positions, the algorithm frequently misclassifies them due to lack of textual or DOM-based identifiers.

These issues highlight the need to incorporate additional visual or semantic analysis methods into future approaches.

Need for standardized benchmarks in E2E web testing. While this paper provides comparability between the different versions of Similo, we still do not assess their performance on actual running web tests, or locators used in practice by existing tests. Indeed, the existing literature on E2E web testing utilizes various benchmarks and evaluation metrics, inhibiting comparisons between tools. Creating a suitable benchmark itself is a challenging task that demands considerable time and effort, which may deter the development of new locator algorithms.

Nevertheless, we advocate the need for the development of a standardized benchmark for evaluating locator generation algorithms, free from the limitations outlined in Section 3.4. Ideally, such a benchmark would encompass a broad spectrum of websites, accommodate frequent updates, and include web elements typical in real-world test scenarios. The availability of a well-curated and consolidated benchmark would enhance the comparability of tools and provide valuable insights into which algorithms are best suited for specific testing environments, potentially leading to the practical application of these locator algorithms if they prove effective against the current state of the art. Recent efforts in this direction include the proposal of the BEWT benchmark by Olianas et al. [27, 28], which represents an initial step toward more realistic and comparable evaluation settings.

7 Related Work

We already discussed Similo [24], VON Similo [23], and LLM VON Similo [25], the first two were included in the empirical comparison conducted in this work. Besides the approaches investigated in this paper, several techniques have been proposed in literature to re-identify changed web elements in E2E web tests [1, 5, 12, 14, 15, 16, 17, 19, 20, 21, 26, 30, 31, 32, 33]. We describe the main propositions next. Similarly to the algorithms investigated in this paper, COLOR [12] re-identifies changed web elements utilizing multiple properties of a target web element. Leotta et al. propose ROBULA+ [17] to create human-readable, short, and simple XPath expressions through an iterative refinement process of a generic XPath until the target element is uniquely identified. In follow-up work, the authors have extended the original algorithm by incorporating attribute robustness ranking, excluding

fragile attributes, and adding textual information for improved stability [20]. In other works, the robust XPath locator problem is formulated as a graph search problem [16, 18]. Montoto et al. [21] propose an algorithm for identifying elements in AJAX websites based on XPath expressions enriched with textual and attribute information.

Concerning ensemble-based strategies, Leotta et al. [19] introduce the novel concept of MultiLocator in which an ensemble of locator-generating algorithms is used to overcome the limitations of individual approaches. The algorithm captures multiple XPath-based locators for each element. It uses them to identify the element by assigning different voting rights or weights to different locators, choosing the one with the highest vote. LLM VON Similo is the latest iteration of the Similo algorithm [25]. The algorithm first uses VON Similo to rank all elements on the website by their score. It then takes the top 10 visual overlaps, as the target element is most likely among them. The algorithm then utilizes GPT-4 [7] to find the target element among the pre-selected candidates. The algorithm converts the ten elements and the target element into JSON format. It then sends a request to the large language model with the target and the ten candidates, asking it to identify the target's match. The LLM responds with a single number, indicating the target's position in the list of candidate elements. Since VON Similo relies on proprietary OpenAI's APIs and uses an unspecified version of GPT-4 with unknown temperature and configuration settings, its results cannot be reliably reproduced in a controlled experimental environment. Therefore, we do not attempt a full replication of this method in this paper. In a recent work, Coppola et al. [6] extend the MultiLocator approach with a Learning to Rank solution to facilitate the relocalization of web elements.

All related studies compare their proposed methods with the state-of-the-art solutions available at the time of publication. In contrast, our work provides an in-depth analysis specifically focused on the family of Similo algorithms, evaluating different versions on both existing and newly developed benchmarks. Our replication study represents a new contribution as it is the first of its kind. Other than the replication, this work provides extensions of the original algorithms in two directions (i.e., a weight-optimized variant and a hybrid approach), other than a larger benchmark.

8 Future Work

To further improve the algorithms accuracy, we believe that extending the set of properties to include non-DOM-based properties would be necessary. Analyzing the results revealed that icons were often mismatched, as the algorithm cannot differentiate between them as effectively as it can with text.

Additionally, future work could address the limitations of using snapshots from the Wayback Machine, such as the reliability and completeness of the data. Exploring alternative sources or methods for obtaining website snapshots, such as utilizing open-source applications with complete version histories, could help mitigate these limitations and provide more accurate and reliable data.

We have observed a correlation between property stability and optimal weights. Finding concrete evidence of correlations between property stability, uniqueness, or other metrics for a specific website could enable a self-tuning and optimizing Similo

algorithm. This would allow the algorithm to adapt to the website's properties and changes over time, further improving its accuracy and robustness.

While this work has focused on reproducibility and providing a locator library that enables replication and reuse of our approach, an important direction for future work is a systematic evaluation of the library or a variation in industrial settings. Such an evaluation could assess its effectiveness on nested, evolving web applications, the effort required to integrate it into existing CI pipelines, and the extent to which developers adopt and save time in practice.

9 Conclusions

This paper replicates two existing studies about web element locator generation algorithms. We discussed the main threats to the validity of the original studies, and re-designed an experimental design to address them. Moreover, our study extends the original studies with a substantially expanded benchmark and configurations.

While our experiments did not entirely replicate the findings of the original study, our results did align with those reported in the replicated studies. However, our findings did not confirm the previously reported superiority of VON Similo over Similo. The discrepancies observed were associated with variations in the benchmark used. In our replication effort, we harmonized the experimentation by utilizing consistent benchmarks.

Our results provide strong justification for the development and adoption of well-curated and standardized benchmarks of webpages, tests and locators for E2E web testing research. Such initiative and endeavor could significantly foster the development of innovative locator generation algorithms by allowing researchers to focus on devising new algorithms rather than on constructing benchmarks. Consequently, this standardization would enable the consistent evaluation of new proposals using consolidated datasets.

10 Declarations

10.1 Funding

This research was funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy.

10.2 Ethical Approval

Not applicable.

10.3 Informed Consent

Not applicable.

10.4 Author Contributions

Anton Kluge: conceptualization, methodology, implementation, evaluation, writing, review, editing. **Andrea Stocco**: conceptualization, methodology, review, editing.

10.5 Data Availability Statement

All our results, the source code, and the library are accessible and can be reproduced [29].

10.6 Conflict of Interest

The authors declare no conflict of interest.

10.7 Clinical Trial Registration

Clinical trial number: Not applicable.

References

1. Amalfitano, D., Coppola, R., Distante, D., Ricca, F.: Ai in gui-based software testing: Insights from a survey with industrial practitioners. In: A. Bertolino, J. Pascoal Faria, P. Lago, L. Semini (eds.) Quality of Information and Communications Technology, pp. 328–343. Springer Nature Switzerland, Cham (2024)
2. Balsam, S., Mishra, D.: Web application testing—challenges and opportunities. Journal of Systems and Software **219**, 112,186 (2025). DOI https://doi.org/10.1016/j.jss.2024.112186. URL https://www.sciencedirect.com/science/article/pii/S0164121224002309
3. Choudhary, S.R., Zhao, D., Versee, H., Orso, A.: Water: Web application test repair. In: Proceedings of the First International Workshop on End-to-End Test Script Engineering, ETSE '11, p. 24–29. Association for Computing Machinery, New York, NY, USA (2011). DOI 10.1145/2002931.2002935. URL https://doi.org/10.1145/2002931.2002935
4. Christophe, L., Stevens, R., De Roover, C., De Meuter, W.: Prevalence and maintenance of automated functional tests for web applications. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 141–150 (2014). DOI 10.1109/ICSME.2014.36
5. Clerissi, D., Leotta, M., Ricca, F.: A gaming quest to improve web locators robustness. In: Proceedings of the 3rd ACM International Workshop on Gamification in Software Development, Verification, and Validation, Gamify 2024, p. 10–17. Association for Computing Machinery, New York, NY, USA (2024). DOI 10.1145/3678869.3685684. URL https://doi.org/10.1145/3678869.3685684

6. Coppola, R., Feldt, R., Nass, M., Alégroth, E.: Ranking approaches for similarity-based web element location. Journal of Systems and Software **222**, 112,286 (2025). DOI https://doi.org/10.1016/j.jss.2024.112286. URL https://www.sciencedirect.com/science/article/pii/S0164121224003303
7. Openai's gpt-4. https://openai.com/research/gpt-4 (2024)
8. Grechanik, M., Xie, Q., Fu, C.: Experimental assessment of manual versus tool-based maintenance of gui-directed test scripts. pp. 9–18 (2009). DOI 10.1109/ICSM.2009.5306345
9. Hammoudi, M., Rothermel, G., Stocco, A.: WATERFALL: An incremental approach for repairing record-replay tests of web applications. In: Proceedings of 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, pp. 751–762. ACM (2016)
10. Hammoudi, M., Rothermel, G., Tonella, P.: Why do record/replay tests of web applications break? In: Proceedings of 9th International Conference on Software Testing, Verification and Validation, ICST 2016, pp. 180–190. IEEE (2016)
11. Indeed.com: Quality assurance engineer salary in united states (2025). URL https://www.indeed.com/career/quality-assurance-engineer/salaries. Accessed: 2025-04-28
12. Kirinuki, H., Tanno, H., Natsukawa, K.: Color: Correct locator recommender for broken test scripts using various clues in web application. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 310–320 (2019). DOI 10.1109/SANER.2019.8667976
13. Leotta, M., Clerissi, D., Ricca, F., Spadaro, C.: Comparing the maintainability of selenium webdriver test suites employing different locators: a case study. In: Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to Testing Automation, JAMAICA 2013, p. 53–58. Association for Computing Machinery, New York, NY, USA (2013). DOI 10.1145/2489280.2489284. URL https://doi.org/10.1145/2489280.2489284
14. Leotta, M., García, B., Ricca, F., Whitehead, J.: Challenges of end-to-end testing with selenium webdriver and how to face them: A survey. In: 2023 IEEE Conference on Software Testing, Verification and Validation (ICST), pp. 339–350 (2023). DOI 10.1109/ICST57152.2023.00039
15. Leotta, M., Ricca, F., Marchetto, A., Olianas, D.: An empirical study to compare three web test automation approaches: Nlp-based, programmable, and capture&replay. Journal of Software: Evolution and Process **36**(5), e2606 (2024). DOI https://doi.org/10.1002/smr.2606. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2606
16. Leotta, M., Ricca, F., Tonella, P.: SIDEREAL: Statistical adaptive generation of robust locators for Web testing. Journal of Software: Testing, Verification and Reliability (STVR) **31** (2021). DOI 10.1002/stvr.1767. URL https://doi.org/10.1002/stvr.1767
17. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: Reducing web test cases aging by means of robust XPath locators. In: Proceedings of 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014, p. (in press). IEEE (2014)
18. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: Meta-heuristic generation of robust xpath locators for web testing. In: 2015 IEEE/ACM 8th International Workshop on Search-Based Software Testing, pp. 36–39 (2015). DOI 10.1109/

SBST.2015.16
19. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: Using multi-locators to increase the robustness of web test cases. In: Proceedings of 8th International Conference on Software Testing, Verification and Validation, ICST 2015, pp. 1–10. IEEE (2015)
20. Leotta, M., Stocco, A., Ricca, F., Tonella, P.: ROBULA+: An algorithm for generating robust XPath locators for web testing. Journal of Software: Evolution and Process pp. 28:177–204 (2016)
21. Montoto, P., Pan, A., Raposo, J., Bellas, F., López, J.: Automated browsing in ajax websites. [Journal Name] **70**(3), 269–283 (2010)
22. Mozilla Developer Network: Css selectors. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_selectors (2024). Accessed: 2024-04-12
23. Nass, M., Alegroth, E., Feldt, R., Coppola, R.: Robust web element identification for evolving applications by considering visual overlaps. In: 2023 IEEE Conference on Software Testing, Verification and Validation (ICST), pp. 258–268. IEEE Computer Society, Los Alamitos, CA, USA (2023). DOI 10.1109/ICST57152.2023.00032. URL https://doi.ieeecomputersociety.org/10.1109/ICST57152.2023.00032
24. Nass, M., Alégroth, E., Feldt, R., Leotta, M., Ricca, F.: Similarity-based web element localization for robust test automation. ACM Trans. Softw. Eng. Methodol. **32**(3) (2023). DOI 10.1145/3571855. URL https://doi.org/10.1145/3571855
25. Nass, M., Alégroth, E., Feldt, R.: Improving web element localization by using a large language model (2024). DOI https://doi.org/10.1002/stvr.1893. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1893. E1893 stvr.1893
26. Nguyen, V.L., To, T.M., Diep, G.H.: Generating and selecting resilient and maintainable locators for web automated testing. Software Testing **31** (2021). URL https://api.semanticscholar.org/CorpusID:233390086
27. Olianas, D., Leotta, M., Ricca, F.: Bewt: A benchmark for end-to-end web testing. In: D. Taibi, D. Smite (eds.) Software Engineering and Advanced Applications, pp. 296–314. Springer Nature Switzerland, Cham (2026)
28. Olianas, D., Leotta, M., Ricca, F.: Bewt: Extended benchmarking for end-to-end web testing. Journal of Systems and Software **237**, 112,849 (2026). DOI https://doi.org/10.1016/j.jss.2026.112849. URL https://www.sciencedirect.com/science/article/pii/S016412122600083X
29. Replication package. https://github.com/ast-fortiss-tum/web-element-localization-using-similo-like-approaches (2025)
30. Ricca, F., Marchetto, A., Stocco, A.: Ai-based test automation: A grey literature analysis. In: 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 263–270 (2021). DOI 10.1109/ICSTW52544.2021.00051
31. Ricca, F., Marchetto, A., Stocco, A.: A retrospective analysis of grey literature for ai-supported test automation. In: J.M. Fernandes, G.H. Travassos, V. Lenarduzzi, X. Li (eds.) Quality of Information and Communications Technology, pp. 90–105. Springer Nature Switzerland, Cham (2023)
32. Ricca, F., Marchetto, A., Stocco, A.: A multi-year grey literature review on ai-assisted test automation (2025). URL https://arxiv.org/abs/2408.06224

33. Ricca, F., Stocco, A.: Web test automation: Insights from the grey literature. In: Proceedings of 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021. Springer (2021)
34. Shijie, H.: Test fragility: An exploratory assessment study on an open-source web application. Master's thesis, Politecnico Di Torio (2019-2020)
35. Similar web. https://www.similarweb.com/top-websites/united-states/ (2024)
36. Stocco, A., Yandrapally, R., Mesbah, A.: Visual web test repair. In: Proceedings of the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), p. 12 pages (2018)
37. Web archive. https://archive.org/ (2024)

A Extended Benchmark

Table 4: Distribution of elements over different websites and locator changes.

	Number of elements			Locator changes		
	Initial elements	Elements available on all versions	Number of element pairs	Same locators	Absolute XPath broke	All locators broke
Accuweather	21	16	262	93	49	120
Adobe	7	6	93	53	9	31
Amazon	42	27	506	247	162	97
Apple	18	17	264	226	4	34
Bestbuy	45	20	469	145	174	150
Bing	10	2	58	40	17	1
Discord	24	15	269	199	16	54
Ebay	45	33	538	260	223	55
Espn	22	13	257	212	20	25
Etsy	36	30	467	244	95	128
Facebook	24	8	203	87	59	57
Fidelity	54	34	727	579	142	6
Google	15	14	211	92	93	26
Indeed	25	19	330	139	107	84
Linkedin	30	17	285	165	0	120
Netflix	16	16	240	137	0	103
Office	41	33	578	290	263	25
Paypal	32	21	377	265	28	84
Reddit	25	2	243	81	61	101
Roblox	31	23	369	222	104	43
T-Mobile	45	27	464	235	135	94
Tiktok	21	1	45	30	2	13
Twitter	28	7	308	248	0	60
Usps	44	35	558	272	283	3
Walmart	49	5	332	99	115	118
Weather	26	19	311	154	85	72
Wikipedia	21	19	295	152	61	82
Yahoo	24	12	287	77	141	69
Zillow	50	20	336	179	43	114
Zoom	62	36	694	449	203	42
Total	**933**	**547**	**10376**	**5671**	**2694**	**2011**

B Full Benchmark

This table shows all results for all algorithms that we evaluated, even those that were not deemed relevant for the main paper.

Table 5: $RQ_{0\text{-}1\text{-}3}$: Results for all algorithms across all benchmarks and evaluation metrics (best results are highlighted in bold).

	M1 (Similo)	M2 (VON)	M3 (LLM)	M4 (Exact)	M5 (LC Exact)	M6 (LC Close)	M7 (Vis. Over.)	M8 (Top Ten)	M9 (Fitness)
Benchmark used for the Similo Paper (809 Element Pairs, 510 with broken locators)									
Similo (Similo)	720 (88.9%)	83.7%	720 (88.9%)	701 (86.6%)	406 (79.6%)	425 (83.3%)	719 (88.8%)	762 (94.2%)	83.1%
Similo (VON)	727 (89.8%)	85.6%	725 (89.6%)	705 (87.1%)	410 (80.4%)	432 (84.7%)	724 (89.5%)	764 (94.4%)	83.8%
VON Similo (VON)	722 (89.2%)	92.8%	722 (89.2%)	627 (77.5%)	353 (69.2%)	432 (84.7%)	704 (87.0%)	773 (95.5%)	76.5%
VON Similo (LLM)	731 (90.3%)	**93.1%**	735 (90.8%)	616 (76.1%)	338 (66.2%)	437 (85.7%)	691 (85.4%)	785 (97.0%)	73.9%
Similo++ (LLM, M4)	758 (93.7%)	75.6%	754 (93.2%)	734 (90.7%)	436 (85.5%)	460 (90.2%)	753 (93.1%)	781 (96.5%)	88.4%
Similo++ (LLM, M3)	760 (93.9%)	83.6%	758 (93.7%)	715 (88.4%)	417 (81.7%)	462 (90.6%)	740 (91.5%)	792 (97.9%)	85.5%
VON Similo++ (LLM, M3)	728 (89.9%)	89.1%	**765 (94.6%)**	540 (66.7%)	299 (58.6%)	446 (87.4%)	626 (77.4%)	792 (97.9%)	65.7%
HybridSimilo (LLM, M4)	**766 (94.7%)**	N/A	764 (94.4%)	**742 (91.7%)**	**443 (86.8%)**	**467 (91.6%)**	**763 (94.3%)**	765 (94.6%)	**89.7%**
Similo++ (Ext, M9)	743 (91.8%)	73.2%	737 (91.1%)	713 (88.1%)	418 (81.9%)	447 (87.6%)	736 (90.9%)	793 (98.0%)	85.4%
VON Similo++ (Ext, M8)	741 (91.6%)	83.3%	738 (91.2%)	635 (78.5%)	358 (70.2%)	447 (87.6%)	718 (88.7%)	**797 (98.5%)**	77.7%
HybridSimilo (Ext, M9)	749 (92.6%)	N/A	745 (92.1%)	719 (88.8%)	424 (83.1%)	453 (88.8%)	743 (91.8%)	**797 (98.5%)**	86.4%
Benchmark used for the VON Similo Paper (441 Element Pairs, 272 with broken locators)									
Similo (Similo)	395 (89.6%)	83.4%	395 (89.6%)	384 (87.1%)	217 (79.8%)	228 (83.8%)	394 (89.3%)	421 (95.4%)	82.4%
Similo (VON)	399 (90.5%)	85.2%	397 (90.0%)	385 (87.3%)	218 (80.1%)	232 (85.3%)	396 (89.8%)	422 (95.7%)	82.8%
VON Similo (VON)	399 (90.5%)	93.6%	397 (90.0%)	323 (73.2%)	173 (63.6%)	235 (86.4%)	379 (85.9%)	428 (97.0%)	71.9%
VON Similo (LLM)	397 (90.0%)	**94.0%**	404 (91.6%)	308 (69.8%)	152 (55.8%)	231 (84.9%)	363 (82.3%)	431 (97.7%)	65.5%
Similo++ (LLM, M4)	415 (94.1%)	76.1%	410 (92.9%)	398 (90.2%)	229 (84.2%)	246 (90.4%)	409 (92.7%)	426 (96.6%)	86.1%
Similo++ (LLM, M3)	416 (94.3%)	84.7%	412 (93.4%)	381 (86.4%)	212 (77.9%)	247 (90.8%)	394 (89.3%)	433 (98.1%)	81.4%
VON Similo++ (LLM, M3)	403 (91.4%)	92.3%	**415 (94.1%)**	254 (57.6%)	126 (46.3%)	238 (87.5%)	314 (71.2%)	433 (98.1%)	54.5%
HybridSimilo (LLM, M4)	**417 (94.5%)**	N/A	414 (93.8%)	**402 (91.1%)**	**233 (85.6%)**	**248 (91.1%)**	**413 (93.6%)**	415 (94.1%)	**88.1%**
Similo++ (Ext, M9)	406 (92.1%)	74.1%	399 (90.5%)	385 (87.3%)	219 (80.5%)	239 (87.8%)	398 (90.2%)	433 (98.1%)	83.3%
VON Similo++ (Ext, M8)	407 (92.3%)	86.1%	404 (91.6%)	323 (73.2%)	172 (63.2%)	241 (88.6%)	384 (87.1%)	**436 (98.8%)**	72.1%
HybridSimilo (Ext, M9)	407 (92.3%)	N/A	402 (91.1%)	386 (87.5%)	220 (80.8%)	240 (88.2%)	400 (90.7%)	**436 (98.8%)**	83.7%
Benchmark used for the LLM VON Similo Paper (803 Element Pairs, 500 with broken locators)									
Similo (Similo)	722 (89.9%)	84.2%	722 (89.9%)	703 (87.5%)	404 (80.8%)	423 (84.6%)	721 (89.8%)	758 (94.4%)	84.2%
Similo (VON)	729 (90.8%)	86.2%	727 (90.5%)	707 (88.0%)	408 (81.6%)	430 (86.0%)	726 (90.4%)	759 (94.5%)	84.9%
VON Similo (VON)	724 (90.1%)	92.9%	724 (90.1%)	629 (78.3%)	351 (70.2%)	430 (86.0%)	706 (87.9%)	770 (95.9%)	77.5%
VON Similo (LLM)	732 (91.1%)	**93.2%**	736 (91.6%)	618 (76.9%)	335 (67.00%)	434 (86.8%)	692 (86.1%)	781 (97.2%)	74.8%
Similo++ (LLM, M4)	760 (94.6%)	75.7%	757 (94.2%)	737 (91.8%)	435 (87.00%)	458 (91.6%)	756 (94.1%)	776 (96.6%)	89.7%
Similo++ (LLM, M3)	762 (94.9%)	84.1%	761 (94.7%)	718 (89.4%)	416 (83.2%)	460 (92.0%)	743 (92.5%)	786 (97.8%)	86.8%
VON Similo++ (LLM, M3)	731 (91.0%)	89.1%	**768 (95.6%)**	544 (67.7%)	298 (59.6%)	445 (89.0%)	629 (78.3%)	789 (98.2%)	66.8%
HybridSimilo (LLM, M4)	**768 (95.6%)**	N/A	767 (95.5%)	**745 (92.8%)**	**442 (88.4%)**	**465 (93.0%)**	**766 (95.4%)**	768 (95.6%)	**91.1%**
Similo++ (Ext, M9)	745 (92.8%)	73.6%	740 (92.1%)	716 (89.1%)	417 (83.4%)	445 (89.0%)	739 (92.0%)	787 (98.00%)	86.7%
VON Similo++ (Ext, M8)	743 (92.5%)	83.9%	741 (92.2%)	638 (79.4%)	357 (71.4%)	445 (89.0%)	721 (89.8%)	**794 (98.8%)**	78.9%
HybridSimilo (Ext, M9)	751 (93.5%)	N/A	748 (93.1%)	722 (89.9%)	423 (84.6%)	451 (90.2%)	746 (92.9%)	**794 (98.8%)**	87.7%
Extended Benchmark (10376 Element Pairs, 2012 with broken locators)									
Similo (Similo)	10276 (99.0%)	89.1%	10275 (99.0%)	10274 (99.0%)	1926 (95.8%)	1928 (95.8%)	10274 (99.0%)	10354 (99.8%)	97.6%
Similo (VON)	10286 (99.1%)	**91.4%**	10285 (99.1%)	10282 (99.1%)	1934 (96.1%)	1938 (96.4%)	10284 (99.1%)	10361 (99.8%)	97.8%
VON Similo (VON)	10168 (97.9%)	90.3%	10270 (98.9%)	9448 (91.0%)	1689 (83.9%)	1934 (96.1%)	10267 (98.9%)	10356 (99.8%)	90.6%
VON Similo (LLM)	10230 (98.6%)	90.4%	10291 (99.1%)	9494 (91.5%)	1681 (83.6%)	1939 (96.4%)	10210 (98.4%)	10357 (99.8%)	90.2%
Similo++ (LLM, M4)	10322 (99.5%)	79.1%	10324 (99.5%)	10321 (99.4%)	1957 (97.3%)	1958 (97.4%)	10323 (99.5%)	10368 (99.9%)	98.6%
Similo++ (LLM, M3)	10328 (99.5%)	85.1%	10322 (99.5%)	10320 (99.4%)	1958 (97.4%)	1966 (97.7%)	10321 (99.4%)	10371 (99.9%)	98.6%
VON Similo++ (LLM, M3)	9779 (94.2%)	90.8%	10308 (99.3%)	8,000 (77.1%)	1415 (70.4%)	1810 (90.00%)	8,705 (83.9%)	10368 (99.9%)	78.1%
HybridSimilo (LLM, M4)	10311 (99.4%)	N/A	10307 (99.3%)	10304 (99.3%)	1940 (96.4%)	1947 (96.8%)	10306 (99.3%)	10315 (99.4%)	98.1%
Similo++ (Ext, M9)	**10356 (99.8%)**	79.7%	**10356 (99.8%)**	**10352 (99.7%)**	**1988 (98.8%)**	**1991 (99.00%)**	**10355 (99.8%)**	10374 (99.9%)	**99.4%**
VON Similo++ (Ext, M8)	10194 (98.2%)	90.6%	10314 (99.4%)	9475 (91.3%)	1699 (84.5%)	1944 (96.6%)	10313 (99.4%)	**10375 (99.9%)**	91.1%
HybridSimilo (Ext, M9)	**10356 (99.8%)**	N/A	**10356 (99.8%)**	**10352 (99.7%)**	**1988 (98.8%)**	**1991 (99.00%)**	**10355 (99.8%)**	**10375 (99.9%)**	**99.4%**

C Cross Validation

We cross validated our results on five temporal folds, training on older snapshots and testing performance on new ones. The results for Similo++ and HybridSimilo for this validation can be found here.

Table 6: Similo++ on the extended dataset with target metric M6: Test set performance across 5-fold temporal cross-validation.

Metric	Mean	Std	Min	Max
M1	99.76%	±0.04	99.71%	99.82%
M2	97.44%	±1.27	95.38%	98.49%
M3	99.76%	±0.04	99.71%	99.82%
M4	99.75%	±0.04	99.71%	99.82%
M5	98.72%	±0.26	98.39%	99.11%
M6	99.33%	±0.12	99.20%	99.54%

Table 7: HybridSimilo on the extended dataset with target metric M6: Test set performance across 5-fold temporal cross-validation.

Metric	Mean	Std	Min	Max
M1	99.67%	±0.11	99.48%	99.77%
M2	97.02%	±2.28	92.58%	98.92%
M3	99.66%	±0.10	99.48%	99.76%
M4	99.64%	±0.09	99.48%	99.73%
M5	98.10%	±0.59	97.07%	98.67%
M6	99.03%	±0.29	98.53%	99.31%