ChatGPT

Open-Source Projects Using diff-dom for Nuanced HTML Diffs

Below we compare several open-source projects that leverage the **diff-dom** library (or similar DOM diffing techniques) to detect and highlight HTML changes. These projects classify diff operations (additions, removals, moves, format changes, etc.) and apply CSS styles to mark changes in rendered HTML, much like Microsoft Word's *Track Changes*. We highlight how each project uses **diff-dom**, handles edge cases (formatting, whitespace, etc.), and any techniques or workarounds that might inform the GAD Inspector's diff engine.

Teamwork's visual-dom-diff (teamwork/visual-dom-diff)

Teamwork's **visual-dom-diff** library produces a merged DOM fragment annotated with `<ins>` and `<del>` tags and CSS classes to denote additions, removals, and modifications [1]. Key aspects of its approach include:

- **Classification of Changes:** Inserted text/content is wrapped in an `<ins>` tag with class `vdd-added` and deletions in a `<del>` tag with class `vdd-removed` [1]. Whole elements added or removed get these classes on their nodes, and text changes are output as a deletion immediately followed by an insertion (old text in `<del>`, new text in `<ins>`) [1]. This makes added blocks or inline insertions clearly highlighted (often styled with underline for `<ins>` and strikethrough for `<del>` in CSS).

- **Formatting & Attribute Changes:** Changes that only affect formatting (e.g. a text segment made bold) are treated as *modifications* rather than pure insertion/deletion. The changed text is wrapped in an `<ins class="vdd-modified">` tag (with the new formatting applied) instead of showing both removal and addition [2]. Similarly, if an element's attributes change, the element is marked with `vdd-modified` but not duplicated [3]. This way, stylistic changes are indicated without duplicating large content. Only the new format/attribute is carried into the diffed output, preserving visual formatting while still flagging that a change occurred [2].

- **Structural Rearrangements:** The diff algorithm prefers to identify node relocations (moves) as such instead of as separate removal/addition events. This aligns with diff-dom's non-destructive philosophy of favoring moves over delete-insert pairs [4]. In practice, that means if a DOM block was moved to a different place, visual-dom-diff will not mark the entire block as removed and re-added, but could (when possible) recognize it as the same block moved (thus minimizing unnecessary highlights). This nuance comes from diff-dom's core behavior which visual-dom-diff builds upon [4].

- **Custom DOM Serialization for Diffs:** Notably, visual-dom-diff uses a custom text diff function that **serializes the DOM to a string**, representing each element as a single Unicode private-use

character [5] [6]. This allows it to run a text diff on the serialized DOM structure. The default `diffText` implementation is case-sensitive and **intelligently merges small adjacent text changes** to produce a more human-friendly result [7]. In effect, this helps avoid highlighting every single character change when a word or phrase is modified – the changes are coalesced into larger tokens when appropriate, reducing noise.

- **Ignoring Specific Nodes:** The library provides options to skip certain nodes or their children during diff. For example, one can specify `skipChildren(node)` or `skipSelf(node)` to ignore differences inside a given element or to treat an element as atomic [8]. This is useful for embedded content or sections that shouldn't be diffed (e.g. an `<iframe>` or a script). Ignored nodes are treated as unchanged formatting elements, preventing spurious diffs from binary or non-text content.

Overall, **visual-dom-diff** directly tackles the challenge of presenting rich-text diffs. By wrapping changes with semantic tags and classes, it enables CSS styling to **underline insertions and strike out deletions** (mirroring Word's revision marks). Format-only changes are handled gracefully by labeling them as modified (carrying over new styling) instead of inserting/removing content [2]. This project does not explicitly mention using the diff-dom *library* as a dependency, but it implements a similar algorithm (and even refers to diff-dom's technique of non-destructive diffs). Its strategy of merging DOM trees into one annotated output and providing hooks for custom diff behavior could inform enhancements to GAD Inspector.

Source: *visual-dom-diff* repository [1] [2] [7]

Matrix/Element (matrix-react-sdk)

The Matrix chat client (Element) uses **diff-dom** in its React SDK to show message edit history with visual diffs. When a user edits a message, Element displays the changes with markup indicating what was removed or added. Key features of their implementation include:

- **Inline Revision Highlighting:** The original and edited message HTML are diffed and rendered such that removed text is wrapped in a `<del>` (displayed with strikethrough) and inserted text in an `<ins>` (displayed with an underline) [9]. This styling choice was introduced explicitly in the project's change log ("strikethrough & underline deletions & insertions" [9]), making it easy for users to see what changed in an edited message. For example, if a word was deleted, it will appear ~~crossed out~~, and new words will appear <ins>underlined</ins>.

- **Diff-DOM for Structure, plus Textual Diffs:** The Element team uses **diff-dom** to compute a DOM diff between the original and edited message content (which may contain HTML formatting like bold, italics, lists, etc.). They found diff-dom's structural diff results useful but augmented it for better inline detail. In practice, they call `dd.diff(originalDom, newDom)` to obtain a list of diff operations (actions with a DOM "route" path) [10]. These operations are then interpreted and applied to produce an HTML string with `<ins>` / `<del>` markers.

- **Fine-Grained Text Diffs with Diff-Match-Patch:** To handle cases where a text node has minor changes (for example, one word changed in a sentence), Element combines diff-dom with Google's *diff-match-patch* algorithm. Diff-dom by default might treat an edited text node as one removal and

one addition of the entire text node. Instead, Element post-processes the diff: when a diff-dom operation indicates a text replacement, they perform a secondary character-level diff (using **diff-match-patch**) to pinpoint the changed substring [11]. This way, only the altered characters or words get wrapped in `<del>`/`<ins>`, rather than marking the entire sentence as deleted and reinserted. This hybrid approach yields a much more nuanced diff visualization – unchanged parts of the text remain normal, while small edits are highlighted within the text. (For instance, changing "color" to "colour" would only underline the inserted "u" and strike out nothing, rather than re-marking the whole word.)

- **Handling Redundant or Trivial Diffs:** The team encountered certain edge cases where diff-dom would return redundant operations. One such issue (tracked as diffDOM issue #90) involved diff-dom outputting a `removeTextElement` immediately followed by an `addTextElement` with identical text in the same location (essentially a no-op change) [12] [13]. This could happen due to subtle HTML structure differences or whitespace. The Element developers implemented a **filter to remove canceling pairs of diff actions** [12] [14]. In the diff result array, if a remove-text and add-text action have the same content and same DOM route, they are detected as canceling out, and both are dropped before rendering the diff. This prevents showing a deletion and insertion of identical text, which would confuse users with needless markup.

- **Ignoring Insignificant Changes:** Relatedly, whitespace-only changes or certain HTML-insignificant differences are normalized to avoid visual noise. For example, Element ensures the HTML is wrapped in a single root node before diffing (to avoid issues with diff-dom needing a common parent) [15], and likely normalizes or strips irrelevant newline differences. The Open Web Components community recommends normalizing whitespace for semantic DOM diffs, and Element's use of diff-dom aligns with that principle (diff-dom itself ignores pure whitespace text nodes by default). By filtering out no-op diffs and focusing on real content changes, Element's diff view remains clean.

- **Applying the Diff to HTML:** Once processed, the diff operations are applied to an HTML string or React DOM. Element's implementation reconstructs an HTML snippet that represents the *new message* but with removed portions wrapped in `<del>` and added portions in `<ins>` (similar to visual-dom-diff's output). Because the diff operations come with precise DOM routes (paths to nodes), Element can insert the markers at the correct positions in the new message. The result is then rendered in the *Edit History* dialog, showing the previous message with revisions highlighted. The CSS in Element ensures `<del>` text is red and struck through, and `<ins>` text is green and underlined (colors depending on theme) – mimicking typical revision highlights.

- **Workarounds and Improvements:** Using diff-dom in a rich-text chat app revealed some quirks that Element addressed. Besides the redundant diff workaround above, they also fixed cases of repeated text in diffs [16] (which were resolved by the same filtering technique and possibly by disabling diff-dom's internal text merging). The introduction of diff-match-patch was a key improvement to get more granular changes. They effectively **turned off diff-dom's own text diffing** (by treating any text change as a full replace) and let diff-match-patch handle the fine detail. Diff-dom's option `valueDiffing:false` can be used for this purpose, ensuring diff-dom doesn't try a character-by-character diff itself [17]. Element's approach shows how diff-dom can be integrated into a larger system: using it for the heavy lifting of structural and block-level diffs, but augmenting it with other algorithms and custom logic for the best visual result.

Source: *Matrix/Element commit history and code* [9] [12] [14] [11]

Fidus Writer's diffDOM (Collaborative Editing)

Fidus Writer, an open-source web word processor, created **diffDOM** (the original diff-dom library) as part of its real-time collaboration system [18] [19]. While not a front-end *visual* diff tool by itself, diffDOM's design and usage in Fidus Writer illustrate strategies for handling complex HTML changes:

- **Non-Destructive Patches:** diffDOM represents differences as a list of operations (a "diff" object) that can be applied to transform one DOM into another [20]. A core principle is that it tries to **preserve existing nodes and structure** where possible. Instead of always treating changes as remove-then-add, diffDOM will identify moved nodes and changes in text content without tearing down the whole structure [20]. For example, if a paragraph was moved to a different section in the document, diffDOM would ideally output a single "move node" operation, rather than a deletion of that paragraph and a separate insertion elsewhere. This preserves object identity and is more semantically meaningful. Fidus Writer found this crucial for collaboration, as it avoids unnecessary content flicker and keeps track of elements like comments or footnotes attached to those nodes.

- **Granular Text Operations with Hooks:** By default, diffDOM treats any changed text node as a replacement (it produces a `removeTextElement` and `addTextElement`). It does **not automatically perform a character-level diff on text** [21]. This is actually by design, to keep the core algorithm simpler and to let users decide how to handle text merges. diffDOM provides hook functions `textDiff` and `textPatch` that developers can override to merge text changes in a custom way [22]. Fidus Writer could use these hooks (or an external diff like diff-match-patch) to intelligently merge concurrent edits or preserve some text formatting. For instance, if two collaborators type in the same paragraph, a naive diff might conflict; with a custom `textDiff`, they could merge the changes character-by-character. In essence, diffDOM's extensibility for text diffs is a notable technique: **separating structural diff logic from text diff logic**, allowing the latter to be customized or replaced.

- **Performance and Large Documents:** In a blog post, Fidus Writer noted an initial performance issue where diffDOM took ~27 seconds to diff a 178 KB HTML document [23]. They optimized the algorithm (e.g. by limiting the search depth or improving the way it matches nodes) and **achieved a diff in 0.3–0.5 seconds for the same document** [24]. This highlights techniques like capping the maximum diff operations or short-circuiting when differences exceed a threshold (diffDOM allows setting a max diffs limit on initialization for debugging or performance [25]). The outcome was a *much faster diff* that enabled real-time collaboration on large documents [26]. These optimizations ensure that even with nuanced handling (like identifying moves or formatting changes), the diff algorithm remains efficient for practical use.

- **Use in Collaboration:** In Fidus Writer's collaborative editing, whenever a user made an edit, the system would take a DOM snapshot *before and after* the change and run diffDOM to get the list of changes [19]. This diff would then be sent to other clients and applied to their DOM (using diffDOM's `apply` method) to update their view. By transmitting diffs (which are typically much smaller than the whole HTML), the system saved bandwidth and applied changes faster. It also meant the diffs had to be robust – for example, purely cosmetic differences (whitespace, or different HTML id

attributes on the same element) should ideally be ignored to avoid "false" diffs. DiffDOM's semantic approach helped here, but it required careful tuning (e.g., possibly ignoring certain attribute changes or classes that don't affect content). The success of diffDOM in Fidus Writer demonstrates that with the right handling of edge cases (like merging text and skipping unimportant changes), even complex structured documents (with tables, lists, etc.) can be diffed and patched in near real-time [27].

Source: *diffDOM library (fiduswriter/diffDOM)* [20] [22] and Fidus Writer blog [19] [24].

Comparison of Techniques and Takeaways

All these projects show **nuanced strategies** for DOM diffs and highlighting, which could benefit GAD Inspector's diff engine:

- **Use of `<ins>`/`<del>` and CSS** – Both visual-dom-diff and Matrix Element wrap changes in semantic tags or add classes so that insertions and deletions can be styled distinctly [1] [9]. This approach cleanly separates content from presentation: the diff output remains valid HTML, and the styling (underline, strikethrough, color) is done via CSS. GAD Inspector can adopt a similar scheme, ensuring that any generated diff markup uses consistent tags/classes for added, removed, and modified content, allowing flexible styling.

- **Categorizing Diff Operations** – Rather than treating every change uniformly, these tools classify differences:

- *Block-level additions/removals:* added or removed nodes get container-level highlights (e.g. an entire new `<div>` might just get a border or background to indicate addition).
- *Inline text changes:* handled at the text node level with ins/del wrapping of only the changed parts.
- *Structural moves:* diffDOM attempts to detect moves; visual-dom-diff and diffDOM both try to avoid labeling moved content as a deletion+insertion [20]. If GAD Inspector's engine can track node identities (keys or stable IDs), highlighting a move could be done with a different annotation (e.g. a special class "moved" or color) rather than two separate changes.

- *Formatting changes:* It's beneficial to treat purely stylistic or attribute changes as a separate category. Visual-dom-diff's strategy of wrapping changed text in an "modified" marker and carrying only the new style [2] is one way to show that, for example, a word was italicized or a link's URL changed, without duplicating the entire content. This suggests GAD Inspector could detect if an element's textual content is the same but formatting differs, and then mark it as a format modification.

- **Merging & Ignoring Trivial Changes** – As seen in Matrix Element's case, diff algorithms sometimes produce extraneous operations (like remove+add of the same text, or multiple small diffs for one logical change). It's important to post-process diffs to merge or drop such cases. Element's **filterCancelingOutDiffs** function is a good example: it scans for a remove followed by an identical add at the same location and removes them [12] [14]. GAD Inspector could implement similar logic to avoid highlighting changes that aren't real changes (e.g. whitespace-only edits or reordering of equivalent attributes). Additionally, normalizing whitespace and line breaks before diffing can prevent noise. The semantic-dom-diff testing tool normalizes whitespace and strips comments for

meaningful comparison [28]; a visual diff tool can do likewise or simply not mark insignificant whitespace differences as edits.

- **Hybrid Diff Approaches** – A powerful lesson is combining structural diff with text diff. DiffDOM focuses on DOM node structure, while algorithms like **diff-match-patch** excel at string diffs. Matrix's approach of using both yields very precise results [11]. GAD Inspector could similarly:

- Use diff-dom (or a similar DOM differ) to get high-level changes.

- For any text nodes that changed, run a finer string diff to highlight exactly what changed inside the text.
  This two-level diff ensures that added or removed blocks are identified, but small edits within a paragraph don't result in the whole paragraph being flagged as replaced. It provides a Word-like experience where only the changed words/letters are colored. DiffDOM even allows a custom text diff function hook [22] – one could integrate diff-match-patch through that hook for a cleaner solution.

- **Skip/Protect Certain Content** – If the HTML may contain sections that should not be diffed (embedded widgets, code blocks, etc.), providing a way to skip them is useful. Visual-dom-diff's `skipSelf/skipChildren` options [8] suggest an implementation: e.g., GAD Inspector could let certain selectors be treated as atomic (no diff within them, even if they differ, perhaps always treat as changed as a whole). This avoids messy diffs inside complex subcomponents and focuses on relevant differences.

- **Performance Considerations** – Handling large HTML with many changes can be slow. Fidus Writer's experience indicates the need for optimizations like limiting diff depth or operations [24]. GAD Inspector might incorporate:

- A threshold to abort diffing if changes exceed a certain count (to avoid hang – diffDOM can be initialized with a max changes limit for safety [25]).
- Using keys or IDs on elements to assist matching (diffDOM uses element IDs as hints to match moved elements [29] [30]). Ensuring that, for example, unique identifiers or stable structural markers are present in the HTML will help the diff algorithm recognize unchanged vs. moved content more accurately, reducing false differences.
- Caching intermediate results if repeatedly diffing similar content, or diffing in chunks if only part of the document changed (as diff-dom-streaming does by feeding a ReadableStream of HTML [31] [32]). While streaming diff is advanced, the idea of incremental diff could be applied if GAD Inspector inspects dynamic content updates.

In summary, these projects underscore the importance of **presenting HTML diffs in a human-friendly way**. By directly using diff-dom for DOM structure and augmenting it for inline changes and special cases, they achieve a track-changes style visualization. Techniques like using semantic tags for highlights, merging adjacent diff operations, and leveraging specialized text diff algorithms can greatly improve the clarity of the diff output. Adopting these strategies will help GAD Inspector produce intuitive diffs – highlighting added blocks, removed lines, and even subtle inline edits – while gracefully handling formatting tweaks and minimizing distraction from insignificant changes.

Sources: Visual DOM Diff docs [1] [2], Matrix/Element release notes and code [9] [12] [11], diffDOM README [20] [22], Fidus Writer blog [19] [24].

---

[1] [2] [3] [5] [6] [7] [8] GitHub - Teamwork/visual-dom-diff: Highlight differences between two DOM trees.
https://github.com/Teamwork/visual-dom-diff

[4] [20] [21] [22] [25] diff-dom - npm
https://www.npmjs.com/package/diff-dom/v/1.0.0

[9] [16] @mtbl/matrix-react-sdk | Yarn
https://classic.yarnpkg.com/en/package/@mtbl/matrix-react-sdk

[10] [11] [12] [13] [14] workaround diff-dom returning redundant diff actions · matrix-org/matrix-react-sdk@95f9e48 · GitHub
https://github.com/matrix-org/matrix-react-sdk/commit/95f9e48f912d700e762839fac11e7ba0f1a69dd5

[15] src/utils/MessageDiffUtils.tsx · v3.42.2-rc.4 · iLab / Pubhubs ... - GitLab
https://gitlab.science.ru.nl/ilab/pubhubs-matrix-react-sdk/-/blob/v3.42.2-rc.4/src/utils/MessageDiffUtils.tsx

[17] fiduswriter/diffDOM: A diff for DOM elements, as client-side ... - GitHub
https://github.com/fiduswriter/diffDOM

[18] [19] [23] [24] [26] [27] 50 times faster collaboration and ProseMirror | Fidus Writer
https://www.fiduswriter.org/2015/10/09/50-times-faster-collaboration-and-prosemirror/

[28] Testing: Semantic Dom Diff: Open Web Components
https://open-wc.org/docs/testing/semantic-dom-diff/

[29] [30] [31] [32] diff-dom-streaming - npm
https://www.npmjs.com/package/diff-dom-streaming/v/0.3.0