Complete diff-dom Operation Taxonomy

diff-dom emits exactly 14 distinct operation types, including 4 operations beyond your current implementation: `modifyValue`, `modifyChecked`, `modifySelected`, and `modifyComment`. These operations use a route-based navigation system with operation-specific properties, and can compose in complex sequences with specific ordering principles that affect rendering in dual-pane diff viewers.

The library implements a **non-destructive diffing philosophy**, preferring node relocation over removal-insertion pairs to preserve DOM node identity and event listeners. All operations are JSON-serializable, reversible through undo operations, and use an index-based route array to navigate from root to target nodes in the original DOM structure.

Complete Operation Set by Hierarchical Category

Text-Level Operations (4 types)

These operations manipulate text nodes and comments, the lowest-level content in the DOM tree.

modifyTextElement - Modifies existing text node content. Structure: `{action, route, oldValue, newValue}`. Triggered when text node data changes between source and target. The node survives with updated content; identity is preserved. For rendering: Left pane highlights `oldValue` text, right pane shows `newValue` with highlight.

addTextElement - Inserts a new text node. Structure: `{action, route, value}`. The route points to the parent node, with the new text inserted at the specified child index. Creates new node identity. Rendering: Left pane shows insertion point indicator, right pane displays `value` with addition highlight.

removeTextElement - Deletes a text node. Structure: `{action, route, value}`. Also updates `parentNode.value` property for textarea elements. Node is destroyed. Rendering: Left pane shows `value` with deletion highlight, right pane shows removal indicator.

modifyComment - Changes HTML comment node content. Structure: `{action, route, oldValue, newValue}`. Rarely emitted in typical HTML diffing but supported for complete DOM coverage. Node identity preserved. Rendering: Show comment syntax with old/new content differentiated by color in respective panes.

Attribute-Level Operations (3 types)

These modify element attributes without changing element structure or content.

addAttribute - Adds new attribute to element. Structure: `{action, route, name, value}`. Element node survives with additional attribute. Rendering: Right pane highlights new attribute name and value in element tag, left pane shows element without this attribute.

modifyAttribute - Changes existing attribute value. Structure: `{action, route, name, oldValue, newValue}`. Special handling: For input elements with `name="value"`, also updates `node.value` DOM property. Element identity preserved. Rendering: Both panes show element with attribute highlighted, displaying old vs new values side-by-side in tag.

removeAttribute - Deletes attribute from element. Structure: `{action, route, name, oldValue}`. Includes `oldValue` for undo capability. Element survives without attribute. Rendering: Left pane highlights removed attribute with strikethrough, right pane shows element tag without it.

Form State Operations (3 types)

These specialized operations track form element state changes, only emitted when `valueDiffing: true` configuration option is enabled (default behavior).

modifyValue - Updates value property of form elements (input, textarea, select, option, progress, param). Structure: `{action, route, oldValue, newValue}`. This captures user input state that may not be reflected in HTML attributes. Element identity preserved, only property changes. Rendering: Show form element with value field highlighted, displaying old vs new input content.

modifyChecked - Toggles checked state of checkbox/radio inputs. Structure: `{action, route, oldValue, newValue}` where values are boolean. Tracks interactive state beyond DOM attributes. Rendering: Show checkbox/radio element with visual checked state indicator (checkmark vs empty) matching left/right values.

modifySelected - Changes selected property of option elements. Structure: `{action, route, oldValue, newValue}` with boolean values. Captures dropdown selection state. Rendering: Highlight option element with selection indicator (checkmark or background color) in respective pane.

Element-Level Operations (3 types)

These add, remove, or replace entire element nodes with their full subtrees.

addElement - Inserts new element with complete subtree. Structure: `{action, route, element}` where `element` is virtual DOM object containing `{nodeName, attributes?, childNodes?, checked?, value?, selected?}`. Creates entirely new node subtree. Rendering: Right pane shows full element with all descendants in addition highlight, left pane shows insertion point indicator or context.

removeElement - Deletes element and entire subtree. Structure: `{action, route, element}` with full element structure for undo. Destroys node and all descendants. Rendering: Left pane shows complete element tree with deletion highlight, right pane indicates removal location with context.

replaceElement - Substitutes one element with completely different element. Structure: `{action, route, oldValue, newValue}` where both values are full virtual DOM objects. This is destructive—original node identity is lost. Typically used when `nodeName` differs or structural compatibility is impossible. Rendering: Side-by-side display of both element trees, left showing `oldValue` structure with deletion styling, right showing `newValue` with addition styling.

Structural Operations (1 type)

relocateGroup - The most complex operation, moving consecutive child nodes to new position non-destructively. Structure: `{action, route, groupLength, from, to}` where `route` identifies parent node, `groupLength` indicates number of consecutive nodes, `from` is source index (can be negative to count from end), and `to` is destination index. This is the library's signature operation, embodying its non-destructive philosophy by preserving node identity and event listeners during movement. Can appear multiple times in sequence affecting the same parent, with indices referencing the original structure before any operations apply.

Edge case: Multiple `relocateGroup` operations with the same route compose into complex reordering sequences. Negative indices in `to` field indicate positions from end of child list. When `from` index exceeds childNodes length during application, crashes occurred in versions 5.0.7+ (Issue #142).

Rendering: Most challenging operation for dual-pane visualization. Show moved nodes highlighted in both panes with connecting flow indicators or matched colors. Left pane displays nodes in original position with "moved from here" indicator, right pane shows same nodes (same identity) at new position with "moved to here" indicator. Consider using animation or visual flow arrows to show relocation path.

Operation Object Structure and Properties

All operations share a base structure with operation-specific extensions:

Universal Properties

action (string) - The operation type identifier, one of the 14 types documented above. Required for all operations.

route (number[]) - Index-based path from root element to target node. Each number represents child index at that tree level. Example: `[1, 0, 3]` means: root's second child (index 1) → first child (index 0) → fourth child (index 3). Used by internal `getFromRoute()` function for DOM navigation. Routes reference the **original DOM structure** before any operations apply, not intermediate states.

Value Properties

oldValue - Previous value for modification and removal operations. Type varies by operation: string for text/attributes, boolean for form states, object for element replacements.

newValue - New value for modification operations. Type matches `oldValue` based on operation type.

value - Value for addition operations (addTextElement, addAttribute). No old/new distinction since adding entirely new content.

Element Properties

element (object) - Complete virtual DOM element representation for addElement/removeElement. Contains `{nodeName, attributes?, childNodes?, checked?, value?, selected?}`. Attributes are key-value string pairs. ChildNodes recursively contain text nodes or element objects.

name (string) - Attribute name for attribute operations (addAttribute, modifyAttribute, removeAttribute).

Relocation Properties

groupLength (number) - Number of consecutive nodes being relocated together. Only for relocateGroup. Can range from 1 (single node) to large numbers for bulk reordering.

from (number) - Source child index in parent. Can be negative to count from end of child list (e.g., -1 for last child). Only for relocateGroup.

to (number) - Destination child index after relocation. Can be negative. Only for relocateGroup.

Virtual DOM Object Schema

Elements use this internal representation:

typescript

```
{
  nodeName: string,           // "div", "span", "#text", "#comment"
  attributes?: {[key: string]: string},
  childNodes?: Array<element>,
  checked?: boolean,          // For checkbox/radio inputs
  value?: string | number,    // For form elements
  selected?: boolean          // For option elements
}
```

Text and comment nodes use simpler structure: `{nodeName: "#text" | "#comment", data: string}`.

Operation Composition and Sequencing

Ordering Principles

Operations compose into arrays that must be applied sequentially. The library follows these ordering patterns:

Text modifications first - `modifyTextElement`, `addTextElement`, `removeTextElement` typically appear before structural changes in the operation array.

Attribute changes interspersed - Attribute operations can appear throughout the sequence, often grouped by affected element.

Relocation clustering - Multiple `relocateGroup` operations affecting the same parent (same route) are batched together. These represent complex reordering of siblings.

Replacements after relocations - `replaceElement` operations typically appear after `relocateGroup` operations, handling nodes that couldn't be matched for relocation.

Form state changes separate - `modifyValue`, `modifyChecked`, `modifySelected` appear as distinct operations even when element structure matches, capturing state beyond markup.

Route Stability Across Operations

Critical principle: **All routes in a diff array reference the original DOM structure**, not intermediate states after applying previous operations. The apply function internally handles index adjustments. This means you cannot simply apply operations one-by-one and use routes literally—the library's apply function manages index shifting caused by insertions, deletions, and relocations.

Cascading Relocation Pattern

Real-world example from Issue #142 showing complex composition:

javascript

```
[
  {action: "modifyTextElement", route: [1,0,1,4], oldValue: "\n   \n   \n   \n ", newValue: "\n   \n "},
  {action: "addTextElement", route: [1,0,1,10], value: "\n   \n "},
  {action: "relocateGroup", groupLength: 1, from: 1, to: 11, route: [1,0,1]},
  {action: "relocateGroup", groupLength: 1, from: 2, to: 13, route: [1,0,1]},
  {action: "relocateGroup", groupLength: 5, from: 3, to: 1, route: [1,0,1]},
  {action: "relocateGroup", groupLength: 1, from: 14, to: 9, route: [1,0,1]},
  {action: "relocateGroup", groupLength: 5, from: 10, to: 4, route: [1,0,1]},
  {action: "relocateGroup", groupLength: 11, from: 9, to: -4, route: [1,0,1]},
  {action: "replaceElement", oldValue: {...}, newValue: {...}, route: [1,0]}
]
```


Pattern insights: Seven consecutive relocateGroup operations target the same parent [1,0,1], representing wholesale reordering of children. Text operations precede structure changes. Final replaceElement affects a different node (parent at [1,0]).

Node Identity Through Operations

Operations affect node identity differently:

Identity preserved: modifyTextElement, modifyAttribute, addAttribute, removeAttribute, relocateGroup, modifyValue, modifyChecked, modifySelected, modifyComment. The node object remains the same JavaScript reference with modified properties.

Identity destroyed: removeElement, removeTextElement. Node and all descendants are removed from tree and dereferenced.

Identity created: addElement, addTextElement. Entirely new node objects constructed.

Identity replaced: replaceElement. Original node destroyed, new node created with different identity even if structurally similar.

This matters for rendering: When identity is preserved, you can visually indicate the same node changed. When replaced, show separate old and new nodes.

Edge Cases and Special Scenarios

Empty Elements

Empty elements (`<div></div>`) transitioning to containing children generate multiple `addElement` operations—one per child. These are NOT batched into a single operation. Each child appears as separate addition at its respective route index.

Identical Elements

When source and target are structurally identical, `diff()` returns an **empty array** `[]`. Your dual-pane renderer should handle this by showing identical content in both panes with "no differences" indicator.

Negative Indices in Relocation

The `from` and `to` properties in `relocateGroup` can be negative integers, counting from end of child list. Example: `to: -4` means "fourth position from the end". This enables efficient specification of moves relative to list end without calculating exact indices.

Boundary issue: When `from` index exceeds actual childNodes length during application, crashes occurred in versions 5.0.7+. The apply function should validate indices or wrap in bounds checking.

Whitespace Text Nodes

HTML whitespace between elements creates text nodes that get diffed. Many operations in real diffs involve whitespace modifications: `oldValue: "\n \n \n"` to `newValue: "\n \n"`. Consider rendering options: show whitespace literally with visible characters (·, ⏎), collapse whitespace display, or hide pure-whitespace text operations entirely based on user preference.

Invalid HTML Normalization

If input HTML is malformed, browsers normalize it via innerHTML parsing. When normalized HTML differs from expected structure, operations may be skipped or unexpected operations generated. The library cannot diff what the browser has restructured.

Form Values vs Attributes

Form element values can change through user interaction without updating HTML attributes. The `valueDiffing: true` option (default) generates `modifyValue`, `modifyChecked`, `modifySelected` operations to capture these state changes. When disabled, only attribute changes are tracked, missing interactive form state.

SVG and XML Case Sensitivity

HTML is case-insensitive by default, but SVG/XML are case-sensitive. Use configuration `caseSensitive: true` when diffing SVG/XML to prevent false positives from case differences. Example: `viewBox` vs `viewbox` are different in SVG.

Multiline Attributes

Attributes spanning multiple lines can cause parsing errors. Example from Issue #127:

html

```
<div data-interchange="[/some/link, (default)],
    [/some/other/link, (medium)]">
```

Error: `'(default)],' is not a valid attribute name`. Sanitize HTML before diffing or handle parse errors gracefully.

Maximum Difference Cap

Configuration option `diffcap: 500` (default) limits maximum differences processed. When exceeded, `diff()` returns false instead of operation array. This prevents performance issues with radically different documents but means your renderer must handle false return value as "too many differences to compute."

Interaction Patterns Matrix

Common Sequences

Element creation with attributes: Single `addElement` operation includes attributes in the element object. No separate `addAttribute` operations needed.

Element modification: Generates separate operations for each change. An element gaining content and attributes produces: `addAttribute` + `addTextElement` operations targeting the same route.

Node movement with content change: Generates `relocateGroup` for movement PLUS `modifyTextElement` or `modifyAttribute` for content changes. Both may target the same node at different route levels.

Element type change: Always generates `replaceElement` since nodeName differs. Cannot modify `div` into `span`—requires replacement.

Impossible Combinations

relocateGroup targeting text nodes: Text nodes are always leaf nodes with no children. relocateGroup targets parents with multiple children. These never combine.

addElement and removeElement with same route: Contradictory—cannot both add and remove at identical location in single diff. Would instead generate `replaceElement`.

modifyAttribute for non-existent attribute: Would generate `addAttribute` instead. Library distinguishes between adding new vs modifying existing.

Multiple operations with identical action and route: Never occurs. Each operation is unique. Multiple changes to same target generate single operation with cumulative effect, or sequence of different operation types.

Sequential Dependencies

relocateGroup order matters: When multiple relocateGroup operations target same parent, they must be applied in array order. Out-of-order application causes incorrect final positions due to index shifts.

Add before modify: When adding element with attributes, single addElement suffices (attributes embedded). No follow-up modifyAttribute needed. But when adding then immediately changing, generates addElement then modifyAttribute sequence.

Remove after relocate: Node relocated then removed generates both operations: relocateGroup (to move) then removeElement (to delete). This seems redundant but can occur when complex diffing decides temporary relocation before recognizing node should be removed.

Rendering Guidance for Dual-Pane Diff Viewer

General Principles for GAD Implementation

Left pane shows source state with operations applied in "before" context. **Right pane shows target state** with operations applied in "after" context. Use color coding: red/deletion for left-side removals, green/addition for right-side additions, yellow/modification for changes appearing in both.

Node identity visualization: When operations preserve identity (modify operations, relocateGroup), consider using matching background colors or unique identifiers in both panes to show "this is the same node, changed." When identity is destroyed/created, show disconnected elements.

Route-based synchronization: Use the route array to establish correspondence between panes. Elements at the same route in original structure should align vertically for easy comparison. As operations are visualized, maintain this alignment even as operations change tree structure.

Operation-Specific Rendering

addAttribute: Left pane shows element tag without the attribute (perhaps with faded placeholder showing where it would be). Right pane shows element tag with new attribute highlighted in green. In tag syntax: `<div class="existing">` vs `<div class="existing" new-attr="value">`.

modifyAttribute: Both panes show element tag with attribute highlighted. Left shows old value in red, right shows new value in green. Consider inline diff within attribute value for long strings. Example: `class="old-class"` vs `class="new-class"`.

removeAttribute: Left pane shows attribute with red strikethrough or red background. Right pane shows element tag with gap or faded placeholder indicating removal. Include tooltip with removed attribute value for reference.

modifyTextElement: Show text content with inline character-level diff. Use text diffing algorithm (Myers diff, word-level diff) to highlight specific character/word changes. Left pane shows oldValue with deletions in red, right shows newValue with additions in green. Unchanged portions remain neutral color.

addTextElement: Left pane shows insertion point with marker (dotted line, insertion caret, or context preview). Right pane shows full text content with green highlight. If significant whitespace, consider showing whitespace characters explicitly.

removeTextElement: Left pane shows text content with red highlight or strikethrough. Right pane shows removal point with marker or context. Display the removed value for user reference.

modifyComment: Render HTML comments visibly (usually hidden in HTML view). Show `<!-- old comment -->` vs `<!-- new comment -->` with standard modification highlighting. Consider toggle to hide comment operations if too numerous.

addElement: Right pane shows complete element subtree with green highlight or border around entire structure. Left pane shows insertion location with context (siblings above/below) and insertion marker. Consider collapsible tree view if element has deep nesting—show root with expand option.

removeElement: Left pane shows complete element subtree with red highlight or border. Right pane shows removal location with context and deletion marker. Include collapsible view for complex removals. Tooltip or side panel can show full removed structure.

replaceElement: Split view showing both complete element subtrees side-by-side. Left pane shows oldValue structure with red theme, right shows newValue structure with green theme. Draw clear visual boundary between old and new. For large replacements, show tree structure comparison with nodes aligned by similarity where possible.

relocateGroup: **Most complex rendering challenge**. Recommended approach:

1. Identify moved nodes by route + from index calculation
2. Assign unique matching color/identifier to the group in both panes
3. Left pane: Show nodes at original position with "moved from here" indicator (arrow pointing right, dashed border, "moved" badge)
4. Right pane: Show same nodes at new position with "moved to here" indicator (arrow pointing from left, dashed border, "moved" badge)
5. Use matching colors or numbers to pair moved groups across panes (e.g., "Move 1" in blue, "Move 2" in purple)
6. Consider visual flow lines connecting left to right, though this can clutter complex diffs
7. For multiple consecutive relocateGroup operations, number them sequentially ("Relocation 1, 2, 3...") so users can understand reordering sequence

When groupLength > 1, bracket or group the multiple nodes visually to show they moved as a unit.

modifyValue: Show form element (input, textarea, select) with value content highlighted. Left displays oldValue in field with red theme, right displays newValue with green theme. Render actual form element visually if possible (input box appearance) rather than just markup text.

modifyChecked: Render checkbox/radio button element. Left pane shows checkbox in oldValue state (checked vs unchecked) with red tint if unchecked or red border. Right pane shows newValue state with green tint if checked. Visual checkmark or empty box clearly indicates state.

modifySelected: For select/option elements, show dropdown with selected option highlighted. Left pane highlights oldValue selection with red, right highlights newValue with green. If part of larger select, show full option list with selection difference emphasized.

Layout Recommendations

Synchronized scrolling: Link scroll positions between left and right panes so users can compare corresponding regions easily.

Line-by-line alignment: Where possible, align corresponding DOM nodes horizontally across panes using same vertical spacing, even when operations cause structure differences.

Minimap overview: Provide thumbnail overview showing density of changes throughout document. Color code change types (red for removals, green for additions, yellow for modifications) in minimap for quick navigation.

Operation filtering: Allow users to filter by operation type. Toggles for "show additions," "show removals," "show modifications," "show relocations" to reduce visual complexity.

Collapsible unchanged sections: When large document regions are identical, collapse them to "No changes (200 lines)" expander to focus attention on differences.

Route display option: Provide developer mode showing route arrays alongside operations for debugging and understanding DOM structure.

Edge Case Handling in UI

Empty diff array: Display "No differences" message with green checkmark. Still render both panes showing identical content for user verification.

Diff cap exceeded (returns false): Show error message "Document differences exceed limit (500+ operations). Cannot compute diff." Offer alternative: render both documents separately without diff highlighting.

Negative indices: Resolve negative indices before rendering by calculating actual positions. Don't show "-4" to users—display resolved position.

Deep nesting: For deeply nested DOM structures, use indented tree view with expand/collapse controls. Show depth level indicators (vertical lines connecting parent to children).

Whitespace operations: Provide toggle option "Show whitespace changes" to hide or display operations affecting pure whitespace text nodes. When shown, render whitespace characters explicitly (· for space, ⏎ for newline, ⇥ for tab).

Implementation Checklist for GAD

Based on this taxonomy, ensure your GAD specification and implementation includes:

1. **Operation handlers for all 14 types**, including the 4 newly identified (modifyValue, modifyChecked, modifySelected, modifyComment)

2. **Route resolution system** that correctly navigates both panes using index arrays
3. **Route stability handling** recognizing routes reference original structure, not post-operation structure
4. **Index adjustment logic** for sequential operation application, especially for relocateGroup sequences
5. **Identity tracking** to visually link same nodes across panes when identity is preserved
6. **Negative index resolution** for relocateGroup operations
7. **Boundary validation** to prevent crashes on invalid from/to indices
8. **Empty diff handling** for identical documents
9. **Diff cap error handling** for massive document differences
10. **Whitespace visibility controls** for text node operations
11. **Form state visualization** for value/checked/selected operations
12. **Collapsible tree views** for complex addElement/removeElement/replaceElement operations
13. **Visual flow indicators** for relocateGroup operations showing movement across panes
14. **Operation filtering UI** to show/hide operation types
15. **Synchronized scrolling** between left and right panes
16. **Color coding system** consistent across all operation types (red for removals, green for additions, yellow for modifications, blue for relocations)