ProvenanceWidgets: A Library of UI Control Elements to Track and Dynamically Overlay Analytic Provenance

Arpit Narechania, Kaustubh Odak, Mennatallah El-Assady, and Alex Endert

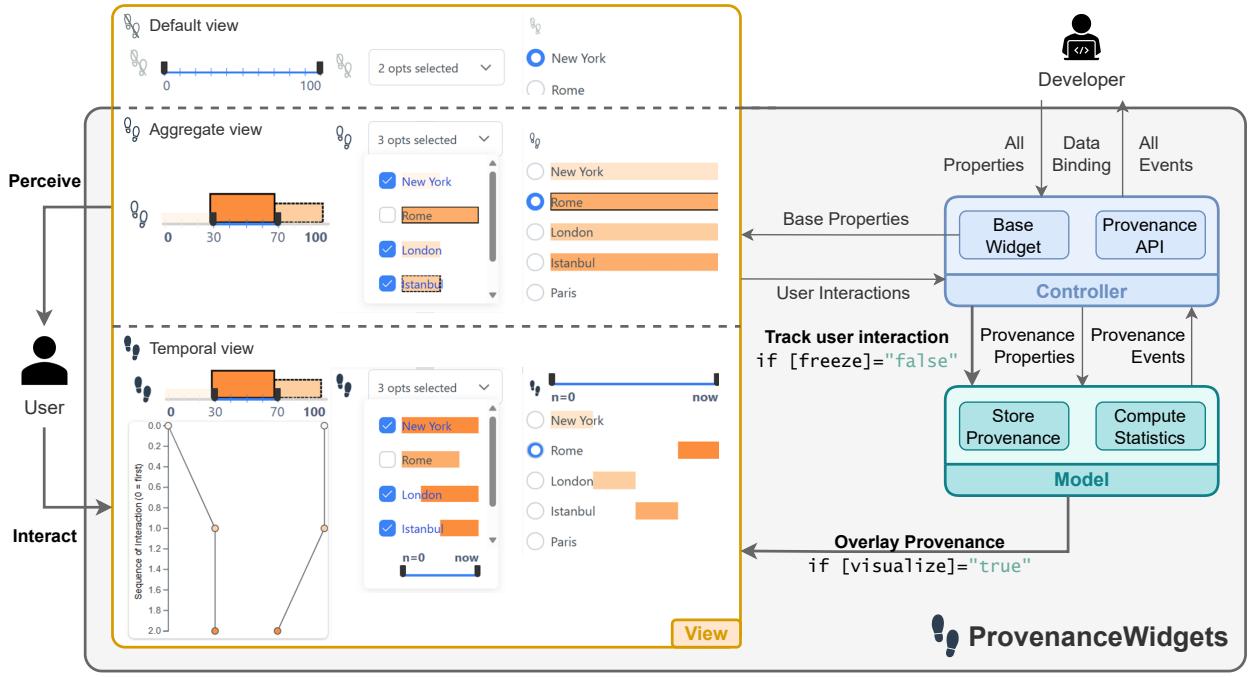

Fig. 1: Overview of ProvenanceWidgets and the underlying Model-View-Controller-based architecture. The Model stores, computes, and updates the provenance. The View shows how end-users perceive and interact with the widgets. The Controller describes how the Model, View, and developers can interact with ProvenanceWidgets.

Abstract—We present ProvenanceWidgets, a Javascript library of UI control elements such as radio buttons, checkboxes, and dropdowns to track and dynamically overlay a user's analytic provenance. These in situ overlays not only save screen space but also minimize the amount of time and effort needed to access the same information from elsewhere in the UI. In this paper, we discuss how we design modular UI control elements to track how often and how recently a user interacts with them and design visual overlays showing an aggregated summary as well as a detailed temporal history. We demonstrate the capability of ProvenanceWidgets by recreating three prior widget libraries: (1) Scented Widgets, (2) Phosphor objects, and (3) Dynamic Query Widgets. We also evaluated its expressiveness and conducted case studies with visualization developers to evaluate its effectiveness. We find that ProvenanceWidgets enables developers to implement custom provenance-tracking applications effectively. ProvenanceWidgets is available as open-source software at `https://github.com/ProvenanceWidgets` to help application developers build custom provenance-based systems.

Index Terms—Provenance, Analytic provenance, Visualization, UI controls, GUI elements, JavaScript library.

1 Introduction

Analytic provenance is the documented history of data and analytical actions, showing how data was obtained, transformed, and analyzed. In a visualization context, analytic provenance tracks how users interact with visualizations as a representation of their reasoning process [55] and can be helpful for recalling the analysis process, reproducing it, collaborating, and logging for evaluation or meta-analysis [59]. Presenting provenance during analysis has been shown to increase awareness of analytic behaviors [50,57], increase confidence [10], reduce exploration biases [76], and result in more unique insights [24,78], among others.

Prior work has made strides in libraries that help developers capture and store provenance [1, 13, 16, 56]. For example, Trrack [16] is an open-source library to create and track the provenance (history) of interactions in web-based applications for a variety of purposes such as action recovery, reproducibility, collaboration, and logging.

While logging and analyzing provenance after analysis has immense value, there is a need for libraries that aid developers in integrating provenance into visual analytic tools in a manner that is consistent with common UI standards. There exist many open-source libraries of UI controls [4, 11, 36, 44, 58, 61, 72, 75] that enhance user interaction and facilitate data input in software applications or websites. Even in visualization and HCI research, there are libraries of enhanced UI controls that facilitate data exploration [30, 73, 79] and navigation [78], explain

- *Arpit Narechania, Kaustubh Odak, and Alex Endert are with Georgia Tech. E-mails: {anarechania3, kodak, endert}@gatech.edu.*
- *Mennatallah El-Assady is with ETH Zürich. E-mail: melassady@ai.ethz.ch.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

transitions in the user interface via afterglow effects [7], or visualize group awareness information [32]. However, there is no frontend library of UI controls that tracks and presents provenance information *out of the box* during analysis. Tools like TrrackVis (Trrack's [16] frontend library) visualize the logged provenance information graph; however, these visualizations are available in a separate view/tool, sometimes after analysis. So we ask: *how can developers integrate provenance directly into the user interface of visual data analysis tools?*

In response, we present **ProvenanceWidgets**, a Javascript library of UI control elements such as radio buttons and dropdowns to track and dynamically overlay a user's analytic provenance, *out of the box*. These enhanced widgets track how often (frequency) and when (recency) a user interacts with them (e.g., selecting a dropdown option) and present visual overlays showing an aggregated summary as well as a detailed temporal history. The aggregated summary is presented in a bar chart overlay visualization encoding the frequency (length) and recency (color) of user interactions with the widget. The detailed temporal history is presented as a timeline visualization, enabling users to access granular information about specific interactions in the past. Presenting provenance as overlays not only saves screen space but also minimizes the amount of time and effort needed to access the same information from other views or external tools.

The ProvenanceWidgets library consists of radio buttons, checkboxes, single sliders, range sliders, dropdowns, multiselects, and input text fields. The library is built using Angular but is universally compatible across different frameworks through Web Components. The library is also highly customizable, allowing developers to realize a variety of configurations such as setting the logging frequency (e.g., logging every interaction or every second), or initializing with a provenance log previously captured.

We demonstrate the capability of ProvenanceWidgets through two usage scenarios and recreate three prior widget libraries: (1) Scented Widgets, (2) Phosphor objects, and (3) Dynamic Query Widgets. In addition, we evaluate the expressiveness of ProvenanceWidgets through cognitive dimensions of notations. We also evaluate the effectiveness of ProvenanceWidgets through case studies with four visualization developers. We find that ProvenanceWidgets enables developers to effectively implement custom provenance-tracking applications. ProvenanceWidgets is available as open-source software at `https://github.com/ProvenanceWidgets` to help visualization developers create new or enhance existing provenance-based systems.

2 Related Work

2.1 Analytic Provenance

Our memory has a limited capacity to remember and track our prior interactions with data, in both amount and decay [42,47], which creates a barrier to data exploration. Analyzing prior interactions with data in visualization is a form of analytic provenance [55,59] that is often used to infer one's analysis process. In fact, provenance has been found to play a crucial role in supporting decision-making, fostering collaboration, and enhancing understanding of complex data analysis in visualization and data analysis contexts [43]. Provenance has also been shown to affect users' confidence levels in conclusions developed, the propensity to repeat work, filtering of data, identification of relevant information, and during typical investigation strategies [10].

Tracking Analytic Provenance. Provenance can be tracked in workflow modeling systems [8] as well as the analysis process within an interactive system [55]. There have been many provenance-logging frameworks [1, 13, 16, 56] that have been used to improve empirical evaluations of visualization techniques [17,54] or teach scientific visualization [68], among others. VisTrails [13] is an open-source system that manages the data and metadata of visualization products by capturing the provenance of both the visualization processes and the data they manipulate. Trrack [16] is an open-source library to capture and replay complete provenance graph; Trrack is complemented by a frontend library (TrrackVis) for visualizing and interacting with this data.

Visualizing Analytic Provenance. Visualizing one's prior interactions can take many forms and lead to shifts in a user's analysis behavior. For instance, when users' prior interactions with charts or data points are encoded (e.g., analogous to coloring previously visited hyperlinks on a webpage purple), people tend to interact with more unique (or previously unvisited) data [24] or the same data repeatedly [50, 76]. Similarly, when exploration history is shown in interactive network visualizations, users report inspiration for conducting further analysis and greater recall of their prior explorations [21]. Scientists have also been found to efficiently and effectively explore their visualizations by returning to previous versions of a dataflow (or visualization pipeline) [13]. Recently, computational notebooks have leveraged provenance information to provide direct feedback on the impact of changes made within cells for iterative and exploratory data analysis [22, 23]. Well-designed histories can also help users maintain contextual awareness of previously visited data when distortions are applied that would otherwise make contextual awareness a challenge (e.g., fisheye lens) [69].

Graphical traces of user interactions have also been utilized in collaborative visualization settings, e.g., to facilitate coordination of multiple users by showing current selections and interactions as "coverage" of the data [6, 63] and in personalized integrated development environments (IDEs), e.g., Footsteps for VSCode [77] highlights the lines of code as the user edits them. Similarly, showing social information "scents" on data visualization widgets (e.g., representing others' interactions with radio buttons, sliders, etc.) leads users to make substantially more unique discoveries in the data [78].

Traces of prior interactions have also been applied in HCI contexts, including tracking user focus while browsing a webpage using eye-tracking [53] and mouse-tracking [5] gear, tracking interactions with documents by authors and readers [33], facilitating groupware coordination [29], revisiting common regions of a page using scrollbar history [2], among others. We refer to Heer et al. [31] for a detailed review of the design space for displaying interaction histories.

2.2 Web and Gaming Analytics

Recently, video game analytics has gained popularity, utilizing game telemetry data [18, 40] to measure player performance [15], characterize player behavior [19, 26, 46] and experiences [18, 35], understand and improve game design [18, 35, 38, 39], and explore social phenomenon [41]. For example, Melo et al. [46] proposed profiling player behavior through provenance graphs and representation learning. Provchastic [39] analyzed game dynamics to predict future game events. Jacob et al. [35] utilized sequential pattern mining algorithms to identify cycles in a game pattern, making the game less tedious and also informing difficulty adjustments. AIRvatar [41] studied how click events and time durations from users' customization of avatars can highlight gender-related stereotypes [41].

Web-based analytics [27, 48, 52] and session replay [25, 34, 49] tools also aid tracking and visualization of user interactions and behavior on websites. These tools capture a variety of metrics such as web page load times, network response times, error rates, including user interactions such as hovers, clicks, and scrolls with the page elements. Beyond low-level data collection, these tools also synthesize high-level user patterns and behaviors that aid performance monitoring (e.g., detect trends and anomalies) and enhance user engagement (e.g., provide guidance). These tools also employ advanced visualizations such as heatmaps to highlight areas of user interaction on web pages, helping optimize their layout and content placement (i.e., facilitate A/B testing) [34, 49].

In contrast, ProvenanceWidgets tracks and visualizes the provenance of user interactions that alter the value of UI controls, independent of broader context like web page load times or game events.

2.3 UI Controls and Libraries

There are many open-source libraries of UI controls that enhance user interaction and facilitate data input in software applications or websites [4, 11, 36, 44, 58, 61, 72, 75]. By utilizing these libraries, developers can expedite their development process while ensuring consistency and accessibility across various platforms and devices.

Visualization and HCI researchers have also developed several enhanced UI control libraries. For example, Phosphor objects [7] instantly show and explain state transitions in GUI controls, e.g., manipulating

a phosphor slider leaves an afterglow that illustrates how the knob moved. Scented Widgets [78] enhance GUI controls with embedded visualizations that facilitate navigation in information spaces. Emotion scents [14] tracks users' emotional reactions while interacting with GUI widgets and visualizes these reactions on the widgets, enhancing the interface for emotional awareness and decision support. TrrackVis provides a customizable provenance visualization front-end for the Trrack library [16]. DynaVis [73] synthesizes persistent UI widgets in response to an initial natural language-based visualization editing task, enabling the user to make subsequent modifications by directly interacting with the widgets (instead of re-typing natural language).

In contrast, ProvenanceWidgets is a library of UI controls with built-in provenance tracking and visualization overlays, and APIs to integrate the provenance information into broader application workflows.

3 Provenance Widgets

3.1 Design Goals

We derived seven design goals based on the goals of prior provenance visualization tools [7, 16, 78] and our own assessment of the capabilities we aim to support in ProvenanceWidgets. Our overarching design goal was to *consistently* achieve the underlying goals for all widgets.

- G1 **Log User Interactions on UI controls as provenance.** The library should automatically track relevant user interactions with the UI controls (e.g., dragging a slider handle) as provenance.
- G2 **Compute Aggregated Metrics about Recency and Frequency of Provenance.** The library should process the logged user interactions and compute aggregate summary metrics pertaining to interaction recency and frequency.
- G3 **Dynamically Overlay Provenance on UI controls.** The library should enhance UI controls with a visual overlay of the aggregate summary metrics and an on-demand temporal evolution of the users' analytic provenance.
- G4 **Support Action Recovery.** The library should allow navigating historical analysis states and also updating the current state, both programmatically and by interacting with the visual overlays.
- G5 **Allow Developer Agency.** Application developers should have the flexibility to tune the default tracking and visualization behavior, including being able to disable it completely. The library should provide an API for the same.
- G6 **Be Framework-Agnostic.** Because multiple frameworks exist, such as Angular, React, and Vue, we derived the goal to design the library to be integrated into any codebase.
- G7 **Support Meta-Analysis.** The library should support logging and exporting provenance information in a format that is suitable for different kinds of analysis. For example, ProvenanceWidgets' internal data structure maintains fine-grained logs as well as higher-level computed aggregates.

3.2 Design Process

As part of our design process, we first reviewed UI controls and then conducted multiple design exercises to decide efficient and consistent visual overlays and associated interactions across all of them.

3.2.1 UI Controls Review

We review the structure, layout, and initial values of radio buttons, checkboxes, single sliders, range sliders, dropdowns, multiselects, and input text fields.

Structure. All aspects of radio buttons, checkboxes, single sliders, and range sliders are completely visible at all times; whereas, dropdowns, multiselects, and input text fields require an additional click and potential scrolling to bring certain aspects (e.g., options) into focus.

Layouts. Dropdowns, multiselects, and input text fields are oriented *horizontally* with their menus opening vertically (above or below depending on screen position); whereas, radio buttons, checkboxes, single sliders, and range sliders can be oriented *vertically or horizontally*.

Initial/Default Values. Radio buttons, checkboxes, dropdowns, multiselects, and input text fields, can have an uninitialized state with *no (null, empty)* selection(s) or value(s); whereas, single sliders and range sliders must always have *at least one* selection.

Subsequent Values. Radio buttons, dropdowns, input text fields, single sliders, and range sliders can have at most *one* selected value; whereas multiselects and checkboxes can have *multiple* selected values.

Interaction Events. Radio buttons, checkboxes, dropdowns, and multiselects are selection-type controls that require the user to *click* to (de)select target options. Sliders and range sliders require the user to *drag* the handle(s) to or directly *click* on the rail at the target value(s). Input text fields generally require the user to first *type* and then *press the 'Enter' key on keyboards* to mark the typing as complete. In ProvenanceWidgets, any interaction event that modifies the *value (or state)* of a widget is logged and included in its provenance computations; an event that does not modify a widget's value, such as *mouseover* or *keyup* is not logged. In addition, clicking on a historical analytic state in the visualization overlays of ProvenanceWidgets is considered a new interaction and is also appended to the widgets' provenance.

3.2.2 Design Exercises and Considerations

We conducted design exercises to explore **what** provenance-related information to show, **where**, **how**, and **when**.

What metrics to log as provenance. We selected two provenance metrics/statistics: frequency and recency of user interactions with widgets (**G2**). We chose these metrics for their relevance to provenance tracking, intuitive comprehension, effective visual encoding, and broad applicability across various domains. Furthermore, these metrics can help derive composite metrics such as durations of different widget states and study interaction patterns within and across widgets.

Where to present provenance. We explored on-demand versus always visible visualizations and considered whether they should be juxtaposed against each other, or overlaid or superimposed on the widgets. Then, we discussed the trade-offs of having separate overlays against pushing surrounding elements away to accommodate the visual provenance scents. Inspired by Shneiderman's Mantra [67], we eventually chose to overlay aggregate views in-place (overview) and temporal views separately on demand considering the level of detail in raw interaction data. We designed a tri-state button that would let us toggle between the different views - default, aggregate, and temporal.

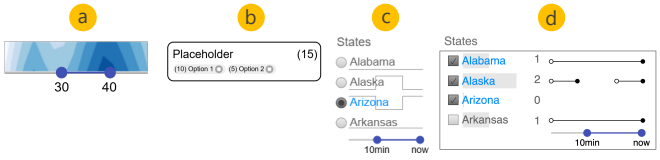

Fig. 2: Alternate designs: range slider, input text, radio button, checkbox.

How to present provenance. We conducted design exercises to explore candidate visualization and interaction techniques to overlay and interact with the logged provenance information. We sketched ideas on draw.io [20] and iterated among co-authors over multiple brainstorming sessions. These low-fidelity sketches included considerations for chart types (e.g., bar charts, line charts, and horizon charts), visual encodings (e.g., color, opacity, size), and UI layouts (e.g., panels, overlays). Keeping in mind our overarching goal of ensuring *consistency*, we selected the *bar* mark and *size, color* encodings to encode frequency and recency information (Figure 4). Figure 2 shows some of our design considerations for sliders, input texts, radio buttons, and checkboxes. For example, we sketched horizon charts in range sliders (Figure 2(a)) and chips in input texts (b), but did not implement them because they did not generalize across all widgets. Similarly, stepped line charts in the temporal view (c) seemed occluding and harder to interact with. We provide all alternate design considerations in supplemental material.

When to log provenance. We considered two kinds of logging frequencies – *interaction-based*, which captures every user interaction when it occurs, and *time-based*, which captures snapshots at specific intervals. Finding utility in both, we chose to support both modes (**G1**, Figure 3).

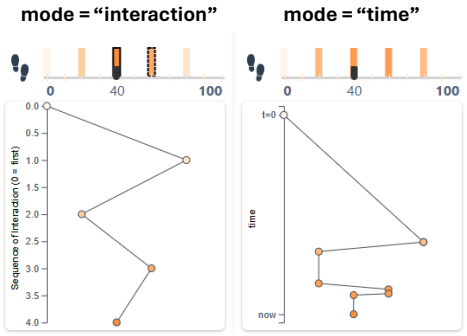

Fig. 3: ProvenanceWidgets: `[mode]="interaction"` and `[mode]="time"` log interactions every interaction and 1 second (by default), respectively.

3.3 Widget Design

Figure 4 shows our chosen designs for the provenance overlays. The default view of each widget is enhanced by the **Aggregate View** (aggregate summary) and a **Temporal View** (detailed history) (**G3**).

When the widget has not been interacted with (i.e., there is no logged provenance), a disabled footprint icon-button is placed next to the UI control. When the widget is interacted with for the first time, this icon-button is enabled, and the widget switches to the Aggregate view, which overlays aggregate provenance information. Clicking the footprint icon toggles between this Aggregate view and the Temporal view, that overlays the temporal history of provenance.

3.3.1 Single Slider, Range Slider

Aggregate View. We designed a bar chart that shows previously selected values (slider) or ranges of values (range slider). The frequency of a selection is encoded by height, and the recency of a selection is encoded by color. Taller, darker bars indicate more frequent and recent interactions, respectively. This bar chart is positioned directly above the slider, as in Scented Widgets [78]. Hovering a bar shows a tooltip with the value, timestamp, frequency, and recency of the selection. Clicking a bar updates the slider to the selected value or range.

Temporal View. To visualize the temporal evolution, we designed a popover that is overlaid above or below the slider. Within this popover, we designed a line chart where time is measured along the y-axis, and the slider itself serves as the x-axis. This line chart has one line for a single slider and two lines for a range slider (one for each handle). The line(s) have circular points that represent the exact selections made over time. These points are also colored by the recency of the selections. Hovering a point shows a tooltip with the corresponding value and time of the selection. In addition, clicking a point updates the slider to the selected value or range (**G4**). Lastly, to facilitate navigation, the y-axis can also be brushed to zoom in on more granular, specific time ranges.

3.3.2 Dropdown, Multiselect, Radio Button, Checkbox

We collectively refer to dropdowns, multiselects, radio buttons, and checkboxes as *selection-type* widgets as they all have a similar visual and interaction design for interacting with provenance information.

Aggregate View. We designed a bar chart and placed it under the options list. An option's selection frequency is encoded by the length of the bar underneath it and the recency is encoded by color. Longer and darker bars indicate higher frequency, recency, respectively. Hovering a bar shows a tooltip with the value, timestamp, frequency, and recency of the selection. Clicking a bar updates the option's selection.

Temporal View. To visualize the temporal evolution, we directly modify the aggregate bar chart unlike that in sliders, where we create a new popover. Each bar represents the time range during which the option was selected. The length of the bar represents the duration of the selection, and the color represents the recency. Longer, darker bars indicate higher frequency, recency, respectively. Hovering a bar shows a tooltip with the corresponding time range. Clicking a bar selects the corresponding option, along with other options that were selected at that point in time. Lastly, to facilitate navigation, there is a horizontal range slider to zoom in on more granular, specific time ranges.

3.3.3 Input Text

Aggregate View. We utilize a dropdown list of previously entered values and visualize provenance as a bar chart underneath each list item. The frequency of an input value is encoded by the length of the bar underneath it, and the recency of a selection is encoded by color. Longer, darker bars indicate higher frequency, recency, respectively. Hovering a list item shows a tooltip with the corresponding timestamp, frequency, and recency of the input value. Clicking a list item updates the text input selection to the corresponding value.

Temporal View. To visualize the temporal evolution, we designed a popover that is overlaid above or below the text input. Within this popover, we designed a vertical timeline chart that shows what text input was searched and when. This timeline has circular points that represent the exact search inputs made over time. These points are also colored by the recency of the input searches. Hovering a point shows a tooltip with the corresponding timestamp of input. Clicking a point updates the current text input selection to the corresponding value.

3.4 Architecture

We broadly define the architecture of ProvenanceWidgets using **MVC** (Model-View-Controller), a software design pattern commonly used to develop GUIs (Figure 1). We describe these as follows:

View: *What the user interacts with* - The View handles all concerns related to the appearance of the widgets, including the base widgets and the overlaid provenance. Internally, we define it almost entirely with Angular templates (HTML) and CSS.

Controller: *What the developer interacts with* - The Controller serves as a hub between the developer, the View, and the Model. Essentially, it wraps the base widget and exposes all of its properties and events, in addition to the ProvenanceWidgets API. It passes on all the base widgets' properties to the View templates, and intercepts all incoming events before re-emitting them for the developers. If not frozen, it relays these events and all provenance-related properties to the Model.

Model: *What we interact with* - The Model stores the raw interaction data received from the Controller, and uses it to compute frequency (how many times a value was input) and recency (how recently a value was input). Once the provenance is updated, the Model can emit it via the Controller as an event for the developers to subscribe to. Then, if visualization is enabled, it updates the View with aggregated summaries of frequency and recency (Aggregate View) or raw temporal history (Temporal View) depending on the active mode.

3.5 Implementation

ProvenanceWidgets is implemented using Angular [3] with an extensible API to support flexibility across different systems. To ensure portability across frameworks (e.g., Vue [74], React [60]), we leverage the WebComponents API (**G6**). Below we describe the main properties (attributes) and events available in the ProvenanceWidgets API.

1. **provenance**: The information recorded and computed by the widget to visualize the provenance of interactions. While each widget has a unique provenance structure, all of them record interactions as an array of objects with the selected/input value and the timestamp of the interaction. This property can be used to initialize, restore, modify, and export (**G7**) the provenance of the widget.
2. **provenanceChange**: An event that is triggered whenever the user interacts with the widget such that its *value* (and hence provenance) changes. For example, *clicking* a radiobutton option or *dragging* a range slider handle constitute a valid event; however, *keyup* or *mouseover* events do not contribute to the provenance.
3. **mode**: This property configures the provenance logging frequency (Figure 3). When 'mode' is set to *"interaction"*, the widget logs every user interaction and accordingly recomputes provenance metrics and updates the subsequent visualizations. When 'mode' is set to

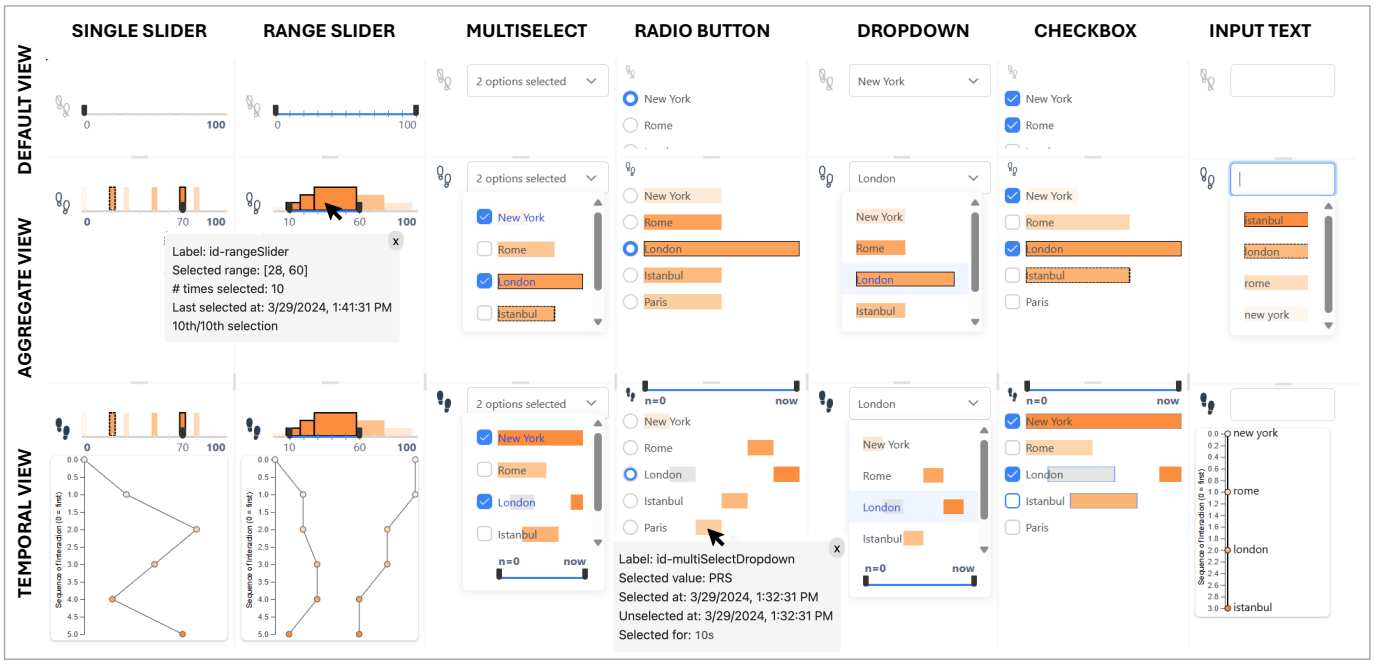

Fig. 4: ProvenanceWidgets: UI controls (single slider, range slider, multiselect, radio button, dropdown, checkbox, and input text) enhanced with an aggregate summary (Aggregate View) as well as a detailed temporal history (Temporal View) of analytic provenance.

"time", the widget logs interactions every 't' seconds (t=1 second by default) and accordingly updates everything downstream.

4. **freeze**: A property to stop logging interactions with the widget. When 'freeze' is set to true, the widget will not record any new interactions, and existing visualizations will not be updated. When 'freeze' is set to false, the widget will continue recording and visualizing the provenance from the last recorded interaction (**G4**).
5. **visualize**: A boolean property to toggle the visibility of the provenance overlays. This property can be used (along with 'freeze'=true) to completely disable provenance in the widget (**G4, G5**).
6. **data-label**: An attribute to pass additional context to the widget. This is displayed as "label" in the widgets' tooltips.

```
provenance?: SliderProvenance | InputTextProvenance |
    Provenance
mode?: "interaction" | "time"
```

```
<provenance-{slider,dropdown,multiselect,radiobutton,
    checkbox,inputtext}
    [(provenance)]="provenance" [mode]="mode"
    [visualize]="true" [freeze]="false"
    [attr.data-label]="'label'"/>
```

Understanding code snippets and notations. We describe the ProvenanceWidgets API using TypeScript and Angular's data binding syntax, which can be categorized based on data flow:

1. *From source to view (property binding).* `[property]="expression"` binds the value from the expression to the property. Can be also used to bind class and style properties, and `data-*` attributes.
2. *From view to source (event binding).* `(event)="function($event)"` executes the bound `function` with the `$event` object emitted by the `event`.
3. *In both ways (two-way binding).* `[(property)]="expression"` binds the value from the expression to the property, and vice versa. It is syntactic sugar for combining property and event binding. For example, `[(provenance)]` is syntactic sugar for `[provenance]` and `(provenanceChange)`.

3.5.1 Single Slider, Range Slider

A slider allows users to select a numeric value from a given range. Traditionally defined as <input type="range"> in HTML, these elements only allow for a single value to be selected. In addition to the traditional sliders, ProvenanceWidgets also provides Range Sliders, which permit selection of a range of values.

```
value: number = 0
highValue?: number = 0 // Omit for Single Slider
handleChange(event: ChangeContext) {
    value = event.value
    highValue = event.highValue }
options: Options = { floor: 0, ceil: 100, step: 1 }
```

```
<provenance-slider [options]="options"
    [value]="value" [highValue]="highValue"
    (selectedChange)="handleChange($event)" />
```

The Slider widget extends SliderComponent from @angular-slider/ngx-slider, and exposes an additional selectedChange event which is triggered at the end of user's interaction with the slider.

3.5.2 Text Input

A Text Input allows users to enter values comprising of text, numbers, and symbols. Traditionally defined as <input type="text"> in HTML, these elements create a basic single-line text input field.

```
value: string = ''
```

```
<provenance-inputtext [(value)]="value" />
```

The Text Input widget extends PrimeNG's AutoComplete component and exposes an additional valueChange event which is triggered when the input value changes.

3.5.3 Dropdown

A Dropdown allows users to select a single value from a list of options. In HTML, these elements are defined using the <select> tag and a list of <option> tags nested within it.

```
type Option = { label: string, value: string }
options: Option[] = []
selected?: Option
```

```
<provenance-dropdown [options]="options"
    optionLabel="label" dataKey="value"
    [(selected)]="selected" />
```

In ProvenanceWidgets, the Dropdown widget extends PrimeNG's Dropdown component and exposes its options attribute to allow developers to provide their list of options.

3.5.4 Multiselect

A Multiselect input allows users to select multiple values from a list of options. In HTML, these are defined in the same way as Dropdowns, but with the multiple attribute set to true on the <select> tag. However, unlike Dropdowns, a Multiselect input renders a list of scrollable options and requires the user to hold down the control key while clicking to select multiple options.

```
selected?: Option[]
```

```
<provenance-multiselect [options]="options"
    optionLabel="label" dataKey="value"
    [(selected)]="selected" />
```

In ProvenanceWidgets, the Multiselect widget extends PrimeNG's MultiSelect component and exposes its options attribute to allow developers to provide their list of options. Unlike a traditional multiselect input, the ProvenanceWidgets Multiselect widget renders in a Dropdown-like manner and does not require users to hold down any keys to select multiple options.

3.5.5 Radio Button

A Radiobutton allows users to select a single value from a list of options. In HTML, these are defined using the <input type="radio"> tag, and all radio buttons with the same name attribute are grouped together.

```
selected?: string
```

```
<provenance-radiobutton [data]="options"
    [(selected)]="selected" />
```

In ProvenanceWidgets, the Radio Button widget extends PrimeNG's RadioButton component. However, unlike traditional Radio Buttons, the ProvenanceWidgets Radio Button widget represents a group of vertically aligned self-contained radio buttons. It exposes a data attribute, which allows developers to provide their list of options instead of having to define each radio button individually.

3.5.6 Checkbox

A Checkbox allows users to select or deselect a single value. Checkboxes can be standalone, or grouped together with the same name attribute. In HTML, these are defined using the <input type="checkbox"> tag.

```
selected ?: string[]
```

```
<provenance-checkbox [data]="options"
    [(selected)]="selected" />
```

In ProvenanceWidgets, the Checkbox widget extends PrimeNG's Checkbox component. Like the Radio Button widget, the ProvenanceWidgets Checkbox widget exposes a data attribute, which allows developers to provide their list of options. All *selection-type* widgets expose a selected attribute, that allows developers to provide an initial selection or override the current selection, and a selectedChange event, triggered when the selection changes.

4 Evaluation

4.1 Example Usage Scenarios

Track Data Transformation and Visualization Specification Interactions (Widget to Visualization). ProvenanceWidgets can help users track what charts they make (visualization specification) and what filters they apply (data transformations). Consider Figure 5 that shows a scatterplot visualization of two attributes: "Year" and "Life Expectancy" along with corresponding single slider and range slider ProvenanceWidgets. As the user drags the slider handle(s): "Year": 1970 → 1990 and "Life Expectancy": [40, 80] → [70.2, 80], the scatterplot updates and also the orange provenance overlays become visible. In this way, the user can utilize ProvenanceWidgets to track already explored data ranges, potentially informing subsequent explorations.

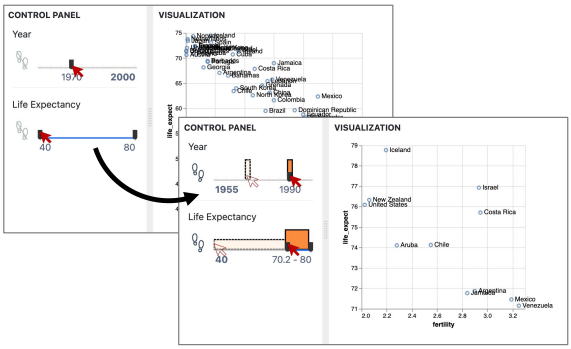

Fig. 5: Visualize Visualization Specifications and Transformation Interactions on ProvenanceWidgets.

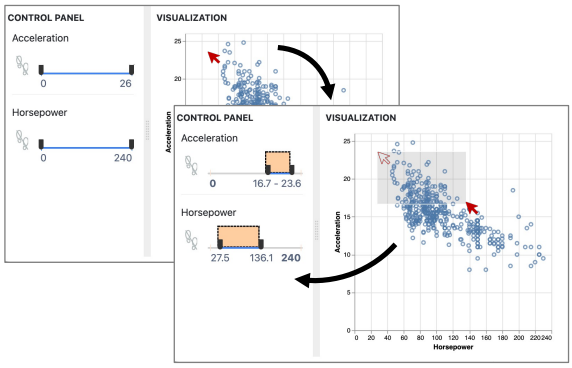

Fig. 6: Track Direct Manipulation-based Interactions in Visualizations.

```
const { view } = await embed("spec.vg.json", ...)
const slider = document.createElement("web-provenance-
    slider");
slider.value = 0;
slider.addEventListener("selectedChange", e => {
    view.signal("slider", e.detail.value).runAsync() })
```

In the above listing, the developer consumes ProvenanceWidgets as Web Components and binds properties and events in JavaScript. They subscribe to "selectedChange" to update the embedded Vega chart [64].

Track Direct Manipulation-based Interactions in Visualizations (Visualization to Widget). Because not all user interactions happen via UI controls, ProvenanceWidgets can be externally updated when user interactions happen elsewhere, e.g., in the visualization.

Consider Figure 6 that shows a scatterplot visualization of two attributes and corresponding range slider ProvenanceWidgets: "Acceleration" and "Horsepower". As the user performs a brush interaction in the visualization, selecting a subset of points within a specific range ("Horsepower": [27.5, 136.1] and "Acceleration": [16.7, 23.6]), the corresponding range sliders can update to show this range. In this way, the user can utilize ProvenanceWidgets to track what data ranges they have already explored, potentially informing subsequent explorations.

```
visBrushed(brush_extent) {
    acc.provenance["revalidate"] = true
    acc.provenance["data"] = [{ ..., "value":
        brush_extent }] // [16.7, 23.6]
}
```

```
<provenance-slider [(provenance)]="acc.provenance" />
```

In the above listing, the developer subscribes to the visualization's brush event via "visBrushed()" and updates the "provenance" of the "Acceleration" ("acc") range slider.

4.2 Replicating Prior Work

We utilize ProvenanceWidgets to replicate three prior works in improving social navigation cues (Scented Widgets [78]), explaining transitions in the user interface (Phosphor Objects [7]), and dynamic

querying-based [66, 79] or direct manipulation-based [30] interactions with a visualization system (Dynamic Query Widgets).

4.2.1 Scented Widgets.

Scented Widgets [78] are graphical user interface controls enhanced with embedded visualizations that facilitate navigation in information spaces. For example, each option in a radio button includes a visual scent of the number of times it has been used across multiple users.

ProvenanceWidgets can be configured to recreate these widgets by showing static information about (2) social navigation, e.g., number of times each radio button option was chosen across multiple users and when (Figure 7A-shades of orange) or (2) data distribution, e.g., distribution of values for that column in the underlying dataset (Figure 7A-blue). To realize the range slider in Figure 7A-shades of orange, the developer can program the widget in the following way:

```
historical_usage_logs = {
  "revalidate": true,
  "data": [{"value": [100, 160], "timestamp": _},
           {"value": [100, 160], "timestamp": _},
           {"value": [160, 200], "timestamp": _}, ...]
}
```

```
<provenance-slider [freeze]="true"
    [(provenance)]="historical_usage_logs" />
```

In the above listing, the developer passes the `"historical_usage_logs"` information in the format of interaction logs, which is then mapped to the `[(provenance)]` property. The `[freeze]="true"` property will ensure the widgets don't update in real-time with user interactions.

4.2.2 Phosphor Objects.

Phosphor objects [7] employ a *phosphor transition* as a transition that (1) shows the outcome of the change instantly via an afterglow effect and (2) also explains the change in retrospect using a diagrammatic depiction. For example, manipulating a phosphor slider leaves an afterglow that illustrates how and from where the knob moved. Users who already understand the transition can continue interacting without delay, while those who are inexperienced or may have been distracted can take time to view the effects at their own pace.

ProvenanceWidgets can be configured to recreate Phosphor objects by limiting the *recency of interaction* mapping to the color encoding channel to just include the two most recent interactions (the current and the previous interaction). That way, every interaction will leave behind a single visual trace (e.g., light green bar) corresponding to the previous value. To realize the single slider in Figure 7B, the developer can program the widget in the following way:

```
widgetUpdated() {
  if (provenance) provenance = {
    "revalidate": true,
    "data": provenance["data"].slice(-2) }
}
```

```
<provenance-slider [(provenance)]="provenance"
    (provenanceChange)="widgetUpdated()" />
```

In the above listing, when a widget is interacted with (`"widgetUpdated()"`), the developer modifies the `"provenance"` array by slicing it to only keep the two most recent interactions and then commanding the library to `revalidate` and recompute its internal model and update the view.

4.2.3 Dynamic Query Widgets.

Dynamic querying. Shneidermann [66] introduced the notion of dynamic queries to continuously update the data that is filtered from the database and visualized. These queries ideally work instantly as the user adjusts UI controls such as sliders or dropdowns to form simple queries or to find patterns or exceptions. Williamson et al. [79] then evaluated this approach in a real-estate system called HomeFinder.

Figure 7C shows how ProvenanceWidgets can create HomeFinder. The developer can program the "Rooms" single slider as follows:

```
// Apply the filter and update the visualization
widgetUpdated(model) {}
```

```
<provenance-slider [visualize]="false" [freeze]="true"
    (selectedChange)="widgetUpdated($event.value)"/>
```

In the above listing, the widget is initialized with `[freeze]`="true" and `[visualize]`="false", which disables logging and hides any overlays. When a widget is interacted with (`"widgetUpdated()"`), the developer can access the new model, filter the data, and update the visualization.

Direct manipulation. Heer et al. [30] introduced direct manipulation techniques that couple declarative selection queries with a query relaxation engine, enabling users to interactively generalize their selections using dynamically generated query widgets. For example, if a user's selections on a housing dataset only include "Home Type"=*Single Family* and "Year"=*2007*, then two dynamic query widgets are created: a checkbox group for "Home Type" with the *Single Family* option checked; and a single slider for "Year", preset to *2007*.

Figure 7C shows how ProvenanceWidgets can support dynamic query widgets created via direct manipulation. To realize the "Year" single slider, the developer can program the widget as follows:

```
selectedYear = 2007;
showWidget = true;
```

```
<provenance-slider *ngIf="showWidget"
    [visualize]="false" [freeze]="true"
    [selected]="selectedYear" />
```

In the above listing, the widget is created (or made visible) by `*ngIf` ="showWidget" and initialized with `[selected]`="selectedYear" (the output of the generalized selection algorithm). The other properties, `[freeze]` and `[visualize]` are still set to "true" and "false", respectively.

4.3 Cognitive Dimensions of Notation

We self-assess our library from a developer standpoint based on the Cognitive Dimensions of Notation [9], a framework of heuristics commonly used to assess the effectiveness of notational systems (e.g., visualization grammars and toolkits). Of the 14 cognitive dimensions, we select a relevant subset for comparing our work with existing tools.

Consistency: *Similar semantics are expressed in similar syntactic forms* – ProvenanceWidgets exposes a common set of provenance-related properties and events, which behave consistently across all underlying widgets (described in Section 3.5). The notation is also consistent with the base libraries it inherits from, as well as consistent across different JavaScript frameworks when used as Web Components.

Diffuseness: *Verbosity of language* and **Hard Mental Operations:** *high demand on cognitive resources* – Since provenance is built into the widgets, developers can directly use components from the underlying libraries to create provenance-aware widgets. Even advanced use cases such as persisting, restoring, or modifying the provenance only require minimal code and cognitive demand. This is in contrast to existing provenance systems such as Trrack and TrrackVis [16], which require developers to set up states, actions, event listeners, and other components to capture and visualize provenance.

Viscosity: *Difficulty of making changes* – The widgets have a low viscosity for primitive attributes, but a high viscosity for complex attributes. For example, developers can easily add labels and toggle provenance tracking and visualization. However, changing the options and provenance data structures requires more effort.

4.4 Expert Case Studies

In this section, we present case studies with developers who utilized ProvenanceWidgets to create custom applications from the ground up. Our aim was to evaluate how effective ProvenanceWidgets is for building provenance-focused visual data analysis systems, as well as to understand developers' experiences working with it, including installation, configuration, and customization.

Participants. We recruited four developers (P_{1-4}) well versed in front-end web development and visual data analysis. Demographically, they were men (2) and women (2) in the 18-24 (1) and 25-34 (3) age groups.

Task. We tasked participants to:

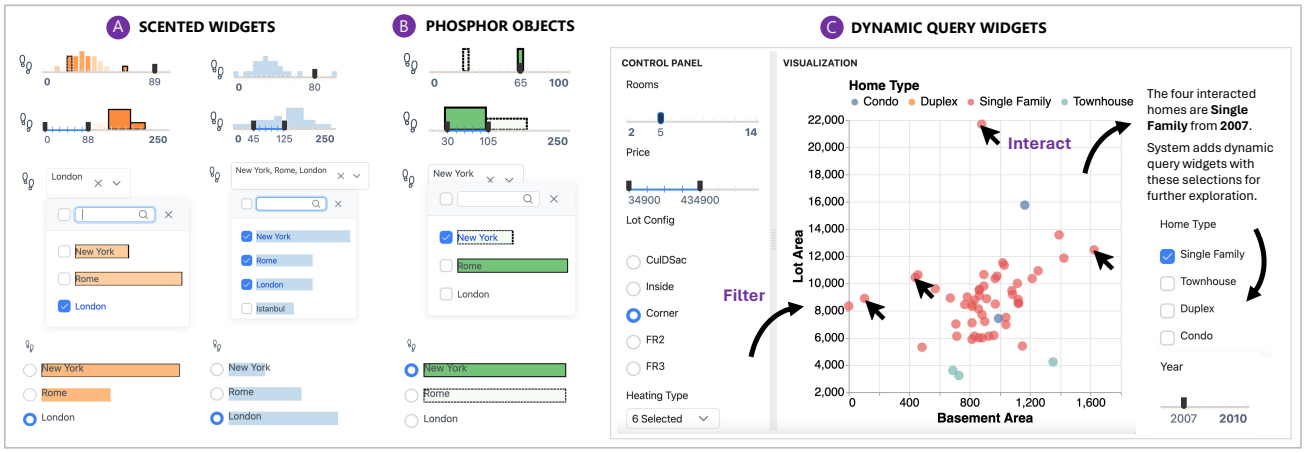

Fig. 7: ProvenanceWidgets can be configured to (re)create the core functionalities of (a) Scented Widgets, (b) Phosphor Objects, and (c) Dynamic Query Widgets. Scented Widgets enhance UI controls via embedded visualizations of some pre-computed metric, e.g., visit frequency and recency (in shades of orange) or data distribution (in blue) to facilitate navigation. Phosphor objects track user interactions with UI controls in real-time and leave visual scents of the most recent (dark green) and second most recent (light green) interaction. Dynamic Query Widgets are UI controls that continuously update a visualization and/or its underlying data as the user adjusts them. For example, ProvenanceWidgets can facilitate creating a dynamic query [66] to lookup affordable ("Price" < $500k) houses with five "Rooms" and "Lot Config"=*Corner* and then update a visualization to see the query result. Alternatively, these widgets can also be created on the fly, e.g., if a user interacts with "Home Type"=*Single Family* and "Year"=*2007* houses, the system can add new query widgets for "Home Type" and "Year" to generalize the user's selection [30] and facilitate future exploration.

> *Develop a "Pokemon Explorer" visualization system for a Pokemon Fan Club, to help member fans visually explore Pokemon names and stats to pick their dream team. The visualization system should consist of a visualization, and UI controls, that help specify the visualization (e.g., map variables to visual encodings) and/or beautify it (e.g., modify font styles and color schemes). The Club wishes to track fans' interaction behaviors as they explore the data, hence you must use ProvenanceWidgets as your UI controls to help track and visualize each user's analytic provenance. In addition, try to capture relevant user interactions from other, non-ProvenanceWidgets places in your application, and manually update ProvenanceWidgets. For example, brushing within a scatterplot visualization should log the brushed extents on either axis and append them to the provenance data structures of the corresponding attribute filters.*

Dataset. We used a dataset of 802 pokemon [65] comprising nine quantitative variables (*Height_m*, *Weight_kg*, *HP*, *Speed*, *Attack*, *Special Attack*, *Defense*, *Special Defense*, *Happiness*), five nominal variables (*Classification*, *Name*, *Primary Type*, *Secondary Type*, *Is Legendary*), and two ordinal variables (*Pokedex Number*, *Generation*). This variety of variables enables developers to use different widgets.

Logistics. We first conducted a 30-minute onboarding interview over Zoom, during which we sought consent from participants and introduced them to ProvenanceWidgets and the study task. We also asked them their preference between the Angular components and the Web-Component versions of the library. Accordingly, we shared a Github repository with them that included installation instructions, API documentation, task instructions, and starter code.

Next, we gave participants up to one week to complete the task. During this week, we asked them to document their experience (e.g. bugs, happy moments) working with the widgets in a FEEDBACK.md file. If and when stuck, we asked participants to create GitHub issues or directly email the study administrators.

Finally, we conducted a 30-minute debriefing interview wherein we reviewed the participants' visualization systems, source code, and feedback notes. We compensated each participant with a $25 gift card.

Analysis. We manually transcribed the audio recordings and feedback, divided them into smaller sections, and applied open coding [12], specifically, constant comparison and theoretical sampling [71]. Next, we briefly describe the participants' applications, development experience, and feedback on the widgets, including future enhancements.

4.4.1 Developed Applications

Figure 8 shows four applications developed by our participants using Angular ($P_{1,4}$) and Web components ($P_{2,3}$). All participants created a scatter plot-based system using ProvenanceWidgets to apply filters ($P_{1,2,3,4}$), specify visual encodings: *xy* ($P_{1,2,4}$), *color* ($P_{1,4}$), and/or adjust styling ($P_{3,4}$). P_2's scatterplot visualized the output of a UMAP [45] dimensionality reduction algorithm that groups more similar pokemon to be closer to each other. P_4 wanted to be able to export the visualizations, hence they provided additional options to configure the font size, point size, and point opacity. Only P_4 attempted the bonus task, to capture user interactions externally (via brushing in the visualization) and manually update the provenance on the relevant widget.

4.4.2 Developer Experience

General Feedback. All four developers found ProvenanceWidgets to be useful, commending its built-in capability to track and visualize provenance. P_1 said, *"I was initially very surprised and actually very excited like, wow, it's very well-made, doesn't really break. That's all you can ask for in any library like this."* P_3 particularly found the widgets to be 'self-explanatory', especially for Angular and Javascript developers. P_4 appreciated the consistent design of the widgets and the ability to externally modify the provenance.

Development Effort. In terms of overall development effort, P_1 took 7 hours whereas $P_{2,3,4}$ took 3-4 hours to set up their visualization system and integrate ProvenanceWidgets. While P_1 found the study to be *"time-consuming"* but *"really fun"*, $P_{2,3,4}$ found the amount of time and effort to be appropriate. P_2 did not find a very steep learning curve and said, *"'Intuitive' will be a very good word to describe it."*

Development Strategies. Participants found the documentation ($P_{1,2,3}$) and starter code ($P_{2,3}$) helpful. P_2 said, *"The sample code to start with was just very helpful because I pretty much just modified it to suit my use-case and it just worked. This single-handedly cut the amount of time I spent in half."* $P_{2,3}$ requested adding more advanced examples in the eventual documentation. P_1 accessed the original PrimeNG and ngx-slider documentations to try and customize the dropdown options and slider options, respectively. $P_{2,4}$ requested alternate widget layouts, e.g., horizontally laid out radio buttons and checkboxes (P_2) and vertically laid out sliders (P_4). These customizations and configurations are currently restricted and unsupported, respectively, because they would conflict with the provenance overlay implementation. A takeaway for us is to acknowledge these limitations in the library documentation and include a roadmap for future features.

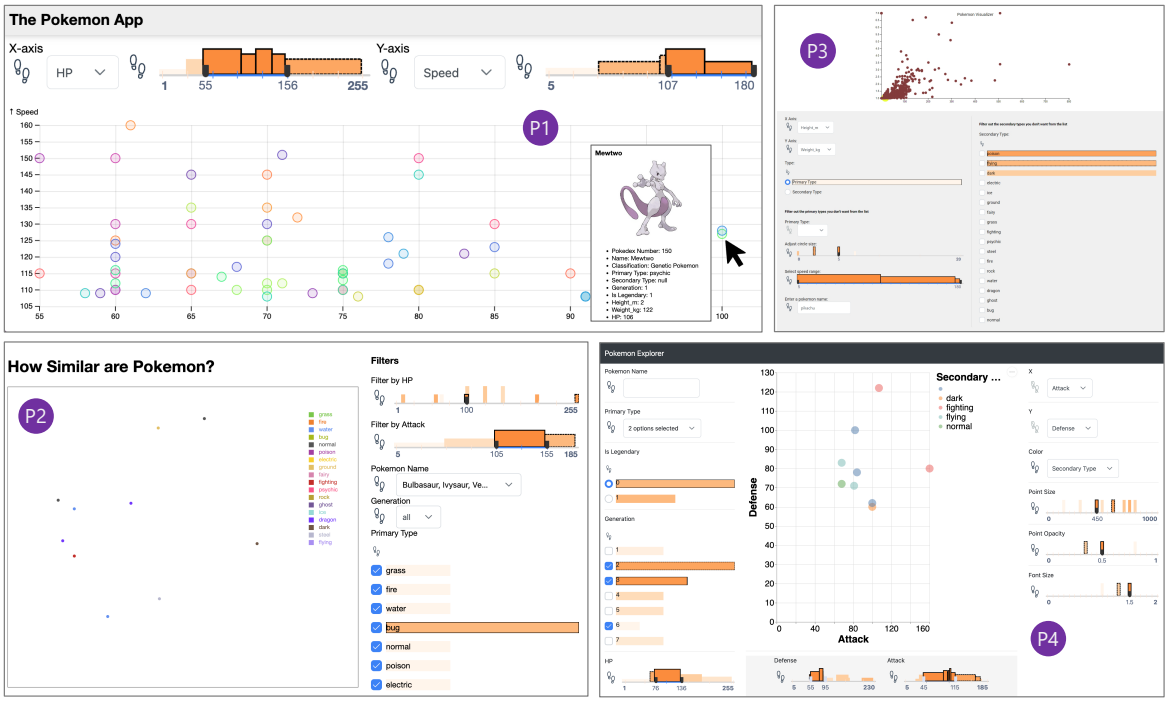

Fig. 8: Pokemon Explorer applications as developed by our developer participants. All participants created a scatter plot-based visualization system using ProvenanceWidgets to apply filters ($P_{1,2,3,4}$), specify visual encodings: *xy* ($P_{1,2,4}$), *color* ($P_{1,4}$), and/or adjust styling ($P_{3,4}$).

4.4.3 Enhancements and Feature Requests.

Our developer participants suggested enhancements to ProvenanceWidgets including additional visualizations ($P_{1,4}$), custom glyphs for checkboxes (P_3), responsiveness to browser window resize events ($P_{1,4}$), better tooltip positioning (P_4), and better support for styling and theming ($P_{2,3}$). P_4 suggested incorporating selective tracking for recency and frequency, and high-precision range sliders for more granular and precise filtering in temporal views. P_4 also suggested enhancements for making the tooltips more human-friendly, such as lowering the precision of numeric attributes (e.g., attack=33.348375), and using 1-based indexing for displaying interactions, instead of '0th/7th interaction'.

5 Discussion

5.1 Limitations and Future Work

ProvenanceWidgets may require developers to understand certain core concepts of its dependencies (PrimeNG, ngx-slider, and Angular), which might lead to issues and limitations, necessitating workarounds. For example, customizing the option templates in the dropdowns and re-orienting the sliders, radio buttons, and checkboxes is currently restricted as it conflicts with the provenance overlays. Future work is planned to ensure ProvenanceWidgets inherits all base library features.

Next, ProvenanceWidgets also inherits the inherent limitations of standard UI controls pertaining to scalability and usability. For instance, sliders and dropdowns often struggle with large ranges and numerous options, respectively. As a workaround, developers can increase a slider's step size (reducing the number of selectable values), improving usability but sacrificing precision; and dropdown options can be reordered or filtered based on frequency or recency to ensure already interacted options are always visible and accessible.

Next, visualizations often involve multi-dimensional interactions like brushing and linking [37] or smart brushing [62], wherein multiple attributes get modified in the same interaction. ProvenanceWidgets can currently visualize such provenance independently on each widget (as demonstrated in Section 4.1, Figure 6), leading to potential misrepresentation and information loss. Future work is planned to track and visualize provenance across multiple widgets.

Lastly, we plan to provide a more flexible and extensible API that will support more widgets, allow customization of existing widgets (e.g., placing provenance scents more freely, e.g., use stroke and opacity instead of color and size), and facilitate creation of new custom widgets (e.g., date pickers). We will also include the ability to compute additional metrics beyond frequency and recency (e.g., their combination).

5.2 Research Opportunities

In this work, we evaluated how visualization developers can configure ProvenanceWidgets, but it remains to be studied how well the widgets can enhance users' visual data analysis workflows. Below we present relevant research opportunities to pursue in the future.

Systematically Studying Analytical Provenance. ProvenanceWidgets can be used to compute advanced metrics to understand co-usage behavior across multiple UI controls, detect biases in control panel usage, and identify over-explored or under-explored aspects of data [50]. Such analyses could provide valuable insights into user behavior and guide interface improvements to mitigate biases and optimize user workflows.

Designing Control Panels. Oftentimes, certain UI controls receive disproportionate usage due to their visibility or positioning. ProvenanceWidgets could offer solutions such as sorting or filtering affordances to re-layout the control panel widgets based on usage statistics. Additionally, visual cues like usage provenance indicators could be implemented to provide users with insights into the popularity of different controls, promoting more uniform interaction behaviors.

Facilitating Co-Adaptive Guidance. ProvenanceWidgets can be extended to facilitate co-adaptive guidance [70], where UI elements dynamically adapt to user interactions in real-time. Such systems can provide personalized assistance while optimizing overall analytical workflows, fostering a more symbiotic relationship with the user.

6 Conclusion

We presented ProvenanceWidgets, a Javascript library of UI control elements to track and dynamically overlay a user's analytic provenance (i.e., the history of their interactions with the UI control elements). Using ProvenanceWidgets, we recreated three prior widget libraries: (1) Scented Widgets, (2) Phosphor objects, and (3) Dynamic Query Widgets. Case studies with four developers revealed the effectiveness of ProvenanceWidgets to build custom provenance-based applications. ProvenanceWidgets is available as open-source software at `https://github.com/ProvenanceWidgets`.

Supplemental Materials

ProvenanceWidgets is available as open-source software at https://github.com/ProvenanceWidgets/ProvenanceWidgets. Supplemental material is available in the IEEE Xplore digital repository as well as Github (https://github.com/ProvenanceWidgets/Supplemental-Material); it includes (1) a video demonstration of ProvenanceWidgets, (2) a design document illustrating the alternative designs we explored during the design and development of ProvenanceWidgets, (3) the application source code demonstrating ProvenanceWidgets replicating prior work, and (4) the detailed information including source code of developer case studies that we used for evaluating ProvenanceWidgets.

Acknowledgments

This material is based upon work supported by NSF IIS-1750474. We are grateful to our user study participants, members of the Georgia Tech Visualization Lab, and anonymous reviewers for their feedback at different stages of this work. We used Google Scholar [28] and vitaLITy [51] to assist us during our literature review.

References

[1] W. Aigner, S. Hoffmann, and A. Rind. Evalbench: A software library for visualization evaluation. In *Computer Graphics Forum*, vol. 32, pp. 41–50. Wiley Online Library, 2013. doi: 10.1111/cgf.12091 1, 2

[2] J. Alexander, A. Cockburn, S. Fitchett, C. Gutwin, and S. Greenberg. Revisiting read wear: analysis, design, and evaluation of a footprints scrollbar. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1665–1674, 2009. doi: 10.1145/1518701.1518957 2

[3] Angular. https://angular.io. Accessed: March 31, 2024. 4

[4] Angular Material. https://material.angular.io, 2024. 1, 2

[5] E. Arroyo, T. Selker, and W. Wei. Usability tool for analysis of web designs using mouse tracks. In *CHI'06 extended abstracts on Human factors in computing systems*, pp. 484–489, 2006. doi: 10.1145/1125451.1125557 2

[6] S. K. Badam, Z. Zeng, E. Wall, A. Endert, and N. Elmqvist. Supporting Team-First Visual Analytics through Group Activity Representations. In *Graphics Interface*, pp. 208–213, 2017. doi: 10.5555/3141475.3141515 2

[7] P. Baudisch, D. Tan, M. Collomb, D. Robbins, K. Hinckley, M. Agrawala, S. Zhao, and G. Ramos. Phosphor: explaining transitions in the user interface using afterglow effects. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pp. 169–178, 2006. doi: 10.1145/1166253.1166280 2, 3, 6, 7

[8] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: Enabling interactive multiple-view visualizations. In *VIS 05. IEEE Visualization, 2005.*, pp. 135–142. IEEE, 2005. doi: 10.1109/VISUAL.2005.1532788 2

[9] A. F. Blackwell, C. Britton, A. Cox, T. R. Green, C. Gurr, G. Kadoda, M. S. Kutar, M. Loomes, C. L. Nehaniv, M. Petre, et al. Cognitive dimensions of notations: Design tools for cognitive technology. In *Cognitive Technology: Instruments of Mind: 4th International Conference, CT 2001 Coventry, UK, August 6–9, 2001 Proceedings*, pp. 325–341. Springer, 2001. doi: 10.1007/3-540-44617-6_31 7

[10] J. E. Block, S. Esmaeili, E. D. Ragan, J. R. Goodall, and G. D. Richardson. The Influence of Visual Provenance Representations on Strategies in a Collaborative Hand-off Data Analysis Scenario. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):1113–1123, 2023. doi: 10.1109/TVCG.2022.3209495 1, 2

[11] Bootstrap. https://getbootstrap.com, 2024. 1, 2

[12] R. E. Boyatzis. *Transforming Qualitative Information: Thematic Analysis and Code Development.* Sage Publications, 1998. 8

[13] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. VisTrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 745–747, 2006. doi: 10.1145/1142473.1142574 1, 2

[14] D. Cernea, C. Weber, A. Ebert, and A. Kerren. Emotion scents: a method of representing user emotions on gui widgets. In *Visualization and Data Analysis 2013*, vol. 8654, pp. 168–181. SPIE, 2013. doi: 10.1117/12.2001261 3

[15] G. K. Chung. Guidelines for the design and implementation of game telemetry for serious games analytics. *Serious games analytics: Methodologies for performance measurement, assessment, and improvement*, pp. 59–79, 2015. doi: 10.1007/978-3-319-05834-4_3 2

[16] Z. Cutler, K. Gadhave, and A. Lex. Trrack: A library for provenance-tracking in web-based visualizations. In *2020 IEEE Visualization Conference (VIS)*, pp. 116–120. IEEE, 2020. doi: 10.1109/VIS47514.2020.00030 1, 2, 3, 7

[17] Y. Ding, J. Wilburn, H. Shrestha, A. Ndlovu, K. Gadhave, C. Nobre, A. Lex, and L. Harrison. reVISit: Supporting Scalable Evaluation of Interactive Visualizations. In *2023 IEEE Visualization and Visual Analytics (VIS)*, pp. 31–35, 2023. doi: 10.1109/VIS54172.2023.00015 2

[18] A. Drachen. Behavioral telemetry in games user research. *Game user experience evaluation*, pp. 135–165, 2015. doi: 10.1007/978-3-319-15985-0_7 2

[19] A. Drachen, M. Seif El-Nasr, and A. Canossa. Game analytics–the basics. *Game analytics: Maximizing the value of player data*, pp. 13–40, 2013. doi: 10.1007/978-1-4471-4769-5 2

[20] Draw.io. https://www.draw.io. Accessed: March 31, 2024. 3

[21] C. Dunne, N. Henry Riche, B. Lee, R. Metoyer, and G. Robertson. GraphTrail: Analyzing large multivariate, heterogeneous networks while supporting exploration history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1663–1672, 2012. doi: 10.1145/2207676.2208293 2

[22] K. Eckelt, K. Gadhave, A. Lex, and M. Streit. Loops: Leveraging Provenance and Visualization to Support Exploratory Data Analysis in Notebooks. *OSF Preprint*, 2023. doi: 10.31219/osf.io/79eyn 2

[23] W. Epperson, D. Jung-Lin Lee, L. Wang, K. Agarwal, A. G. Parameswaran, D. Moritz, and A. Perer. Leveraging analysis history for improved in situ visualization recommendation. In *Computer Graphics Forum*, vol. 41, pp. 145–155. Wiley Online Library, 2022. doi: 10.1111/cgf.14529 2

[24] M. Feng, C. Deng, E. M. Peck, and L. Harrison. Hindsight: Encouraging exploration through direct encoding of personal interaction history. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):351–360, 2017. doi: 10.1109/TVCG.2016.2599058 1, 2

[25] FullStory. https://www.fullstory.com. Accessed: June 15, 2024. 2

[26] A. R. Gagné, M. S. El-Nasr, and C. D. Shaw. A deeper look at the use of telemetry for analysis of player behavior in rts games. In *International Conference on Entertainment Computing*, pp. 247–257. Springer, 2011. doi: 10.1007/978-3-642-24500-8_26 2

[27] Google Analytics. https://marketingplatform.google.com/about/analytics. Accessed: June 15, 2024. 2

[28] Google Scholar. https://scholar.google.com, 2024. 10

[29] C. Gutwin. Traces: Visualizing the immediate past to support group interaction. In *Graphics interface*, pp. 43–50. Citeseer, 2002. doi: 10.20380/GI2002.06 2

[30] J. Heer, M. Agrawala, and W. Willett. Generalized selection via interactive query relaxation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 959–968, 2008. doi: 10.1145/1357054.1357203 1, 7, 8

[31] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189–1196, 2008. doi: 10.1109/TVCG.2008.137 2

[32] J. Hill and C. Gutwin. Awareness support in a groupware widget toolkit. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pp. 258–267, 2003. doi: 10.1145/958160.958201 2

[33] W. C. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless. Edit wear and read wear. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 3–9, 1992. doi: 10.1145/142750.142751 2

[34] Hotjar. https://www.hotjar.com. Accessed: June 15, 2024. 2

[35] L. Jacob, E. Clua, and D. de Oliveira. Oh Gosh!! Why is this game so hard? Identifying cycle patterns in 2D platform games using provenance data. *Entertainment Computing*, 19:65–81, 2017. doi: 10.1016/j.entcom.2016.12.002 2

[36] jQuery UI. https://jqueryui.com, 2024. 1, 2

[37] D. A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002. doi: 10.1109/2945.981847 9

[38] T. C. Kohwalter, F. M. de Azeredo Figueira, E. A. de Lima Serdeiro, J. R. da Silva Junior, L. G. P. Murta, and E. W. G. Clua. Understanding game sessions through provenance. *Entertainment Computing*, 27:110–127, 2018. doi: 10.1016/j.entcom.2018.05.001 2

[39] T. C. Kohwalter, L. G. Murta, and E. W. Clua. Provchastic: Understanding

and predicting game events using provenance. In *International Conference on Entertainment Computing*, pp. 90–103. Springer, 2020. doi: 10.1007/978-3-030-65736-9_7 2

[40] T. C. Kohwalter, L. G. P. Murta, and E. W. G. Clua. Capturing game telemetry with provenance. In *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pp. 66–75. IEEE, 2017. doi: 10.1109/SBGames.2017.00016 2

[41] C.-U. Lim and D. F. Harrell. Toward telemetry-driven analytics for understanding players and their avatars in videogames. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, pp. 1175–1180, 2015. doi: 10.1145/2702613.2732783 2

[42] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2122–2131, 2014. doi: 10.1109/TVCG.2014.2346452 2

[43] K. Madanagopal, E. D. Ragan, and P. Benjamin. Analytic provenance in practice: The role of provenance in real-world visualization and data analysis environments. *IEEE Computer Graphics and Applications*, 39(6):30–45, 2019. doi: 10.1109/MCG.2019.2933419 2

[44] Material-UI. https://material-ui.com, 2024. 1, 2

[45] L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018. doi: 10.21105/joss.00861 8

[46] S. A. Melo, T. C. Kohwalter, E. Clua, A. Paes, and L. Murta. Player behavior profiling through provenance graphs and representation learning. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*, pp. 1–11, 2020. doi: 10.1145/3402942.3402961 2

[47] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 101(2):343, 1994. doi: 10.1037/h0043158 2

[48] Mixpanel. https://mixpanel.com. Accessed: June 15, 2024. 2

[49] Mouseflow. https://mouseflow.com. Accessed: June 15, 2024. 2

[50] A. Narechania, A. Coscia, E. Wall, and A. Endert. Lumos: Increasing Awareness of Analytic Behavior during Visual Data Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):1009–1018, 2022. doi: 10.1109/TVCG.2021.3114827 1, 2, 9

[51] A. Narechania, A. Karduni, R. Wesslen, and E. Wall. vitaLITy: Promoting Serendipitous Discovery of Academic Literature with Transformers & Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):486–496, 2022. doi: 10.1109/TVCG.2021.3114820 10

[52] New Relic. https://newrelic.com. Accessed: June 15, 2024. 2

[53] J. Nielsen and K. Pernice. *Eyetracking web usability*. New Riders, 2010. 2

[54] C. Nobre, D. Wootton, Z. Cutler, L. Harrison, H. Pfister, and A. Lex. reVISit: Looking under the hood of interactive visualization studies. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–13, 2021. doi: 10.1145/3411764.3445382 2

[55] C. North, R. Chang, A. Endert, W. Dou, R. May, B. Pike, and G. Fink. Analytic provenance: process+ interaction+ insight. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pp. 33–36. 2011. doi: 10.1145/1979742.1979570 1, 2

[56] M. Okoe and R. Jianu. Graphunit: Evaluating interactive graph visualizations using crowdsourcing. In *Computer Graphics Forum*, vol. 34, pp. 451–460. Wiley Online Library, 2015. doi: 10.1111/cgf.12657 1, 2

[57] J. R. Paden, A. Narechania, and A. Endert. BiasBuzz: Combining Visual Guidance with Haptic Feedback to Increase Awareness of Analytic Behavior during Visual Data Analysis. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pp. 1–7, 2024. doi: 10.1145/3613905.3651064 1

[58] PrimeNG. https://www.primefaces.org/primeng, 2024. 1, 2

[59] E. D. Ragan, A. Endert, J. Sanyal, and J. Chen. Characterizing provenance in visualization and data analysis: an organizational framework of provenance types and purposes. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):31–40, 2016. doi: 10.1109/TVCG.2015.2467551 1, 2

[60] React. https://reactjs.org. Accessed: March 31, 2024. 4

[61] React Bootstrap. https://react-bootstrap.github.io, 2024. 1, 2

[62] R. C. Roberts, R. S. Laramee, G. A. Smith, P. Brookes, and T. D'Cruze. Smart brushing for parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics*, 25(3):1575–1590, 2019. doi: 10.1109/TVCG.2018.2808969 9

[63] A. Sarvghad and M. Tory. Exploiting analysis history to support collaborative data analysis. In *Proceedings of the 41st Graphics Interface Conference*, pp. 123–130, 2015. 2

[64] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017. doi: 10.1109/TVCG.2016.2599030 6

[65] Serebii.net. http://serebii.net. Accessed: March 31, 2024. 8

[66] B. Shneiderman. Dynamic queries for visual information seeking. *IEEE software*, 11(6):70–77, 1994. doi: 10.1109/52.329404 7, 8

[67] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *The craft of information visualization*, pp. 364–371. Elsevier, 2003. doi: 10.1109/VL.1996.545307 3

[68] C. T. Silva, E. Anderson, E. Santos, and J. Freire. Using vistrails and provenance for teaching scientific visualization. In *Computer Graphics Forum*, vol. 30, pp. 75–84. Wiley Online Library, 2011. doi: 10.1111/j.1467-8659.2010.01830.x 2

[69] A. Skopik and C. Gutwin. Improving revisitation in fisheye views with visit wear. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 771–780, 2005. doi: 10.1145/1054972.1055079 2

[70] F. Sperrle, A. Jeitler, J. Bernard, D. Keim, and M. El-Assady. Co-adaptive visual data analysis and guidance processes. *Computers & Graphics*, 100:93–105, 2021. doi: 10.1016/j.cag.2021.06.016 9

[71] A. Strauss and J. Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, 1998. doi: 10.4135/9781452230153 8

[72] SvelteKit UI. https://kit.svelte.dev, 2024. 1, 2

[73] P. Vaithilingam, E. L. Glassman, J. P. Inala, and C. Wang. DynaVis: Dynamically Synthesized UI Widgets for Visualization Editing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, article no. 985, 17 pages. Association for Computing Machinery, New York, NY, USA, 2024. doi: 10.1145/3613904.3642639 1, 3

[74] Vue.js. https://vuejs.org. Accessed: March 31, 2024. 4

[75] Vuetify. https://vuetifyjs.com, 2024. 1, 2

[76] E. Wall, A. Narechania, A. Coscia, J. Paden, and A. Endert. Left, Right, and Gender: Exploring Interaction Traces to Mitigate Human Biases. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):966–975, 2022. doi: 10.1109/TVCG.2021.3114862 1, 2

[77] A. Wattenberger. Footsteps for VS Code. https://marketplace.visualstudio.com/items?itemName=Wattenberger.footsteps, 2021. 2

[78] W. Willett, J. Heer, and M. Agrawala. Scented Widgets: Improving navigation cues with embedded visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1129–1136, 2007. doi: 10.1109/TVCG.2007.70589 1, 2, 3, 4, 6, 7

[79] C. Williamson and B. Shneiderman. The Dynamic HomeFinder: Evaluating dynamic queries in a real-estate information exploration system. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 338–346, 1992. doi: 10.1145/133160.133216 1, 7