Olli with Hypergraphs: An Extensible Architecture for Accessible Charts and Diagrams

Jonathan Zong*
University of Colorado Boulder

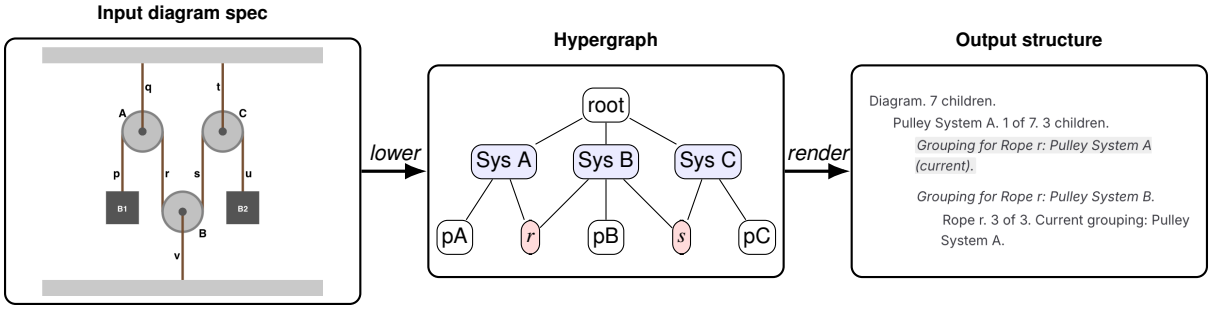

Figure 1: Olli lowers a diagram or chart specification to a hypergraph and renders the hypergraph as a keyboard-navigable tree.

Abstract

Olli is an open-source library that renders data visualizations as keyboard-navigable structured textual descriptions for screen reader users. Olli has represented a chart as a tree derived from the chart's visual encodings; however, two systems have critiqued the restriction to trees. Data Navigator observed that hierarchical trees cannot express relations such as spatial adjacency or membership in multiple groups. Benthic argued that hypergraphs avoid the limitations of trees and graphs because a hypergraph can consistently express both hierarchy and adjacency. In response, this paper presents a new architecture for Olli that uses hypergraphs as its core internal representation. Domain plugins for charts and diagrams lower their specifications to the hypergraph, and a shared navigation runtime traverses the hypergraph. The new architecture unifies Olli with Benthic, and Olli's customizable description and interactive filtering now apply to charts and diagrams alike. Authors can specify Olli structures at three levels of abstraction: Vega-Lite and Bluefish specifications via adapters, Olli's own domain specifications, or raw hyperedges. Navigation on the shared runtime behaves consistently across domains but fits each graphic less closely than navigation authored specifically for that graphic. We invite the community to build new domains and to conduct empirical studies on Olli's shared runtime.

Index Terms: Accessibility, screen readers, data visualization, diagrams, visualization toolkits.

1 Introduction

This paper presents a rearchitecture of Olli[1] [1] around hypergraphs, in response to two critiques of its tree-based navigation. Olli converts a visualization specification into a keyboard-navigable tree view with descriptions at multiple levels of detail. Since its release, Olli has gained research-based features including customizable description [7] and predicate-based filtering [16]. Throughout, Olli has represented a chart as a tree derived from the chart's visual encodings, with axes and legends as branches. Two systems have since critiqued the restriction to trees (Section 2.2). Data Navigator [4] observed that hierarchical trees cannot express relations including spatial adjacency. Benthic [8] argued that neither trees nor graphs match the perceptual structure conveyed visually by a graphic.

Olli with hypergraphs lowers specifications for charts and diagrams to a hypergraph, a graph in which an edge can connect multiple vertices. This hypergraph is the input to a single navigation runtime that manages all user interaction for both kinds of graphic (Figure 1). When the hypergraph is tree-shaped, the runtime offers Olli's existing hierarchical structure and navigation. When an edge has multiple parents, the user can open an extra level that lists the edge's parents, and the user can pick which parent to continue under. The extra level keeps the tree navigation model while supporting a broader set of hypergraph structures.

This architecture enables three capabilities beyond the original Olli and Benthic. First, one runtime controls navigation for both charts and diagrams, so Olli's customizable description [7] and predicate filtering [16] now apply to diagrams. Second, extensible domains enable authors to support new kinds of graphics by implementing a lowerer to the hypergraph (Section 3.4). New domains inherit Olli's runtime, description, and filtering instead of requiring a separate system. We demonstrate this with a diagram domain and a Bluefish [9] adapter. Third, authors can specify structures at three levels of abstraction: a toolkit specification through an adapter, Olli's domain specifications, or raw hyperedges. The Umwelt editor [16] now additionally supports accessible structured editing of Olli visualization specs (Section 3.5).

We discuss the tradeoff between navigation that behaves consistently across graphics and navigation fitted to each graphic, outline potential future work on representations that support both structural and spatial navigation, and invite the community to build extensions to Olli and use it in empirical work on structure and navigation.

2 Related Work

2.1 Structured Navigation of Accessible Visualizations

A line of work in accessible visualization replaces static alt text with interactive, structured access to charts. Zong et al. [15] identified structure, navigation, and description as design dimensions of

*e-mail: jzong@colorado.edu

[1]Olli is open source at https://github.com/umwelt-data/olli, with documentation and interactive examples at https://umwelt-data.github.io/olli/.

screen reader visualization experiences, and evaluated these dimensions through co-design prototypes with blind participants. Olli [1] implements the dimensions as an extensible library with adapters for common visualization toolkits. Chart Reader [11] pairs structured chart navigation with author-curated insights, while VizAbility [5] layers LLM-based conversational query on top of Olli's tree navigation. Commercial systems such as the Highcharts accessibility module [6] also expose charts to keyboard and screen reader interaction. Like the original Olli, most of these systems restrict navigation to a hierarchy derived from a chart's encodings. Data Navigator [4] and Benthic [8] remove that restriction.

2.2 Different Approaches to Structure

Data Navigator [4] observed that lists and trees have expressive limitations; for example, a tree must duplicate any element that belongs to more than one group. Data Navigator's toolkit therefore lets developers construct dynamic graph structures with arbitrary edges. Developers also author the navigation itself through rules that define input bindings and behavior, so each graphic gets a bespoke navigation model.

Benthic [8] argued that a reader perceives two kinds of relation in a graphic, hierarchy and adjacency, and that trees and graphs each express only one kind. Each node of a tree has exactly one parent, so a tree reflects hierarchy but must duplicate any element with a cross-cutting association such as a label or an annotation. A graph avoids the duplication, because any two elements can share an edge. But each edge joins one pair, and an edge does not differentiate between whether it links a group to a member or links two unrelated elements. Benthic instead proposed hypergraphs. A hyperedge connects one parent to a set of children, and children of the same hyperedge are neighbors. Two elements are neighbors exactly when they share a parent, so hierarchy and adjacency stay consistent no matter which edges the author specifies.

3 Lowering Domain Specifications into a Shared Hypergraph

Olli's new architecture follows a compiler's organization, in which many specification languages *lower* (translate) into one intermediate representation and share a runtime. Olli implements compilation in four stages that connect an input specification to the screen reader output:

1. An adapter converts an external format such as Vega-Lite or Bluefish into one of Olli's domain specifications.
2. A domain's lowerer converts the domain specification into a hypergraph.
3. The navigation runtime manages the user's focus and selection over the hypergraph and assembles the description for each node.
4. A renderer presents the runtime's state as an ARIA tree view.

The hypergraph, the runtime, and description assembly form a domain-agnostic core. A domain plugin supplies a lowerer and optional extensions such as description tokens and dialogs (Section 3.4).

3.1 Hypergraphs

Olli lowers every input to a `Hypergraph`, a directed acyclic graph in which an edge may have multiple parents. Acyclicity gives every edge a well-defined path from a root, and the runtime derives the edge's depth and data selection from this path (Section 3.2). Each edge stores a display name, a semantic role such as `root`, `xAxis`, or `element`, lists of parent and child edges, and a domain-specific payload (Section 3.4). A tree is the special case in which every edge has one parent, so a hypergraph can recreate any structure that was previously possible in Olli without changes.

Unlike a tree or a graph, a hypergraph lets one element belong to several groups at once (Figure 2). In the pulley diagram, rope *r* belongs to pulley systems A and B and also connects pulley A to pulley B. A tree would have a rope *r* node under each system, so a user who visits both systems encounters two seemingly unrelated copies. A graph shares rope *r* as one node, but its edges make rope *r*'s group membership indistinguishable from its adjacencies. The hypergraph makes rope *r* one element with two parent hyperedges, so it is shared without a copy and its membership stays distinct from its connections. Figure 3 shows the full hypergraph of a Benthic pulley example.

3.2 The Navigation Runtime and Renderer

The navigation runtime takes a hypergraph as input, manages the user's focus and selection over the hypergraph, and passes a tree of navigation nodes to the renderer. We present every hypergraph to users as a tree, so users navigate every structure with the same interaction model and keep the overview-first progressive disclosure of the original Olli.

Arrow keys drive traversal. The Down arrow key enters a node's children (the last-visited child when the user returns), Up returns to the parent, and Left and Right move across siblings.

If a user encounters a multi-parent edge, pressing Up opens an extra level that lists the edge's parents. The runtime synthesizes this level on demand rather than storing it in the tree. Each entry announces one parent ("Grouping: System A (current)"), and the current parent appears first. Left and Right move across the parents, Up commits to the focused parent, and Down returns to the node as a child of that parent. Benthic calls this operation parent context switching [8]. Adding an extra level for multiple parents fits parent context switching into Olli's existing arrow-key navigation model.

The runtime also assigns every node a selection, a predicate over the data that drives the data table and synchronized visual highlighting. Because selections use Vega-Lite's field predicate types, an embedding application such as Umwelt [16] can observe focus and selection through Olli's handle API and highlight the matching marks in a live chart. When the application pushes a new selection into Olli, the runtime re-lowers the specification and restores focus to the equivalent node.

3.3 Customizable Descriptions

Olli assembles each node's description from a set of *tokens*, which are fragments of text that describe a specific attribute of the node. Tokens are defined by the core and by each domain (Section 3.4). A *recipe* is an ordered list of tokens that describes one semantic role, so for example an axis node and a data point node have different recipes. A *preset* assigns a default recipe to every role and reconfigures all descriptions at once. A user can further customize descriptions by editing recipes in a dialog menu. This design implements Jones et al.'s [7] finding that screen reader users want control over the presence, order, and verbosity of description content.

3.4 Extensible Domains

A *domain* is a plugin that packages the behavior specific to one kind of graphic as an implementation of a shared TypeScript interface. Each domain must implement a lowerer from the domain's specification to a hypergraph. A domain can additionally implement five optional extensions: description tokens, verbosity presets, keyboard shortcuts, dialogs, and predicate providers. The domain interface separates the behavior shared by every navigable graphic from behaviors that are specific to one kind of graphic (like a chart or a diagram).

Olli now has two built-in domains: visualizations and diagrams. The visualization domain contributes nine description tokens, three verbosity presets, shortcuts that jump to axes and legends, and four dialogs (data table, filter, targeted navigation, description settings).

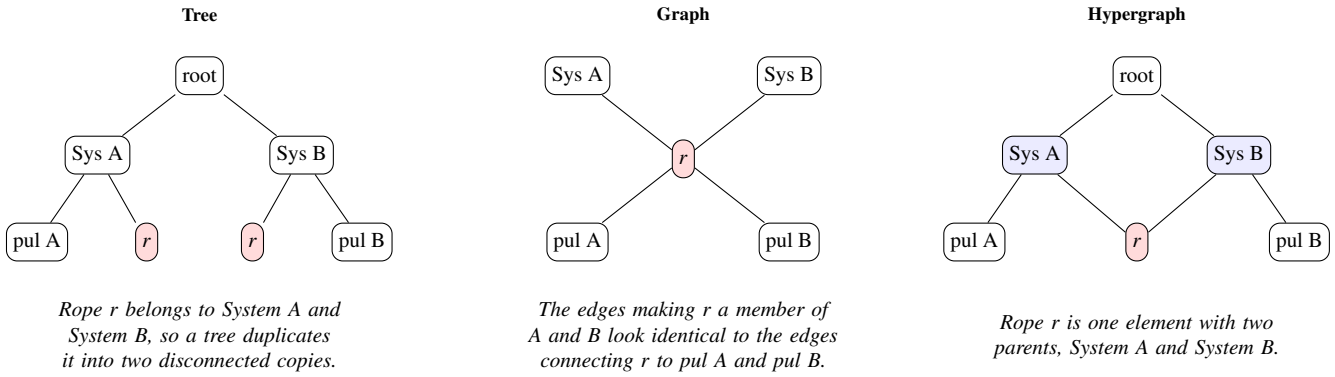

Figure 2: Three representations of the pulley diagram, in which rope r belongs to pulley system A and pulley system B and connects pulley A to pulley B. The tree gives every node one parent, so it duplicates rope r into two disconnected copies. The general graph shares rope r as one node, but every edge means the same thing, so the edges that make rope r a member of the two systems are indistinguishable from the edges that connect it to the two pulleys. In the hypergraph panel, System A and System B are hyperedges that group their children (pul A, rope r, pul B). Rope r is one element with two parent hyperedges, so it belongs to both systems without a duplicate node.

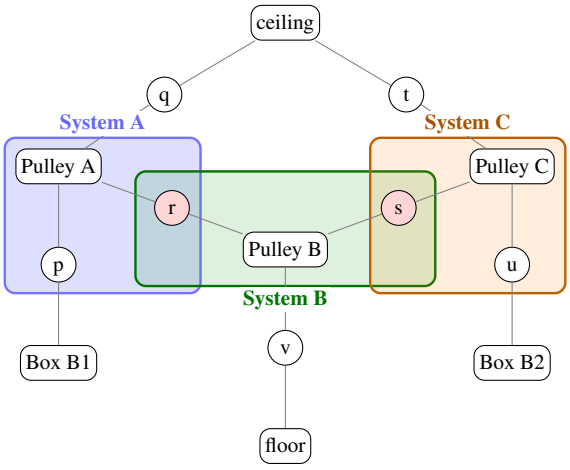

Figure 3: The complete pulley hypergraph from Olli's gallery. The three pulley systems are the shaded groups, and they overlap when two systems share a rope.

The diagram domain contributes one token and one dialog. The token, `elementKind`, announces an element's kind, such as "pulley" or "rope". The dialog is the description settings dialog configured for the seven diagram roles (root, element, connection, containment, alignment, distribution, and grouping), so diagram users get the same customization options as chart users.

The diagram domain and the Bluefish adapter unify Olli with Benthic, which proposed a Bluefish adapter as future work [8]. Together, the adapter and the diagram domain implement Benthic's hypergraph structure and parent context switching, adapted to Olli's keyboard bindings (Section 3.2).

3.5 Authoring at Three Levels of Abstraction

Olli enables authors to specify navigable structure in multiple ways. Authors write specifications at one of three entry points, which span automatic generation and manual authoring.

1. **Toolkit specifications via adapters.** Authors with an existing Vega-Lite or Bluefish specification pass it to an adapter, which produces a domain specification.
2. **Domain specifications.** Authors can write Olli's domain specifications directly. `OlliVisSpec` describes a visualization as fields, encodings, and data. `DiagramSpec` describes a diagram as elements plus five relation types (connection, containment, alignment, distribution, grouping). Within a visualization specification, the `structure` property also gives authors direct control of the navigation hierarchy: a tree of nodes describing *groupby* operations, predicate filters, and annotation highlights, without writing hyperedges. Umwelt [16] allows users to edit this structure in its multi-modal editor.
3. **Raw hyperedges.** An author can write a hypergraph directly and pass it to Olli, with full control over structure, roles, and domain payloads.

4 Discussion and Future Work

Reflecting on structure across systems. Olli, Benthic [8], and Data Navigator [4] differ in three characteristics of navigable structure: which component defines the structure (its source), which shape the structure can take (its kind), and how a user moves through the structure (its navigation model).

The structure's source affects how each system defines correspondence between structure and the graphic it represents. Olli's visualization domain derives the structure from a chart's encodings, so the structure matches the chart's axes and legends. Olli's diagram domain derives the structure from Bluefish's Gestalt relations through the Bluefish adapter, so its structure matches the diagram's perceptual organization. In Data Navigator, and at Olli's raw-hyperedge entry point, a developer hand-authors the structure, so it matches whatever the developer specifies.

The kind of structure (list, tree, or hypergraph) determines which relations a system can express (Section 3.1). The original Olli expressed hierarchy but not adjacency. Data Navigator expresses arbitrary relations, but authors can encode hierarchy and adjacency inconsistently. Olli with hypergraphs expresses both hierarchy and adjacency in one structure.

The navigation model trades fit against consistency. Data Navigator lets developers author navigation per graphic, so navigation can fit each graphic exactly. However, a user can encounter inconsistent navigation across hand-authored graphics. Olli applies the same hierarchical traversal to every hypergraph, so navigation behaves consistently across graphics. But imposing tree navigation on a hypergraph has costs. The synthesized parent level adds a navigation state, and participants in Benthic's study sometimes found the parent context layer confusing [8]. Bespoke navigation could also meet expectations that Olli's generic navigation model cannot.

Benthic's participants, for example, expected to be able to spatially navigate the ropes and pulleys.

Toward mixed structural and spatial navigation. A hypergraph captures discrete grouping and adjacency but not spatial layout. Zong et al. [15] distinguished structural navigation, which moves along an accessible structure, from spatial navigation, which moves along the coordinate directions of the graphic. While hypergraphs are helpful for representing perceptually congruent structure, they do not straightforwardly support spatial navigation. In Benthic's evaluation, participants navigated the hierarchy and adjacency of a pulley diagram yet were unsure which pulleys hung above others and where the floor and ceiling were [8].

Is it possible to create a hybrid representation that expresses both semantic structure (including containment and adjacency) and spatial layout? Recent work on the GoFish visualization grammar indicates a promising direction for exploration. GoFish [10], a declarative visualization grammar from the same Bluefish lineage as Benthic, represents spatial information as part of its specification. GoFish organizes a graphic into three spaces: data, underlying, and display. Its underlying space records semantic spatial information, such as whether an axis is ordinal or continuous, whether an angular axis wraps so that its first and last positions are adjacent, and whether a quantity is a position or an extent. The graphical operators GoFish uses to place marks—stacking, connection, and containment—are the Gestalt relations Benthic encodes as hyperedges, so a single GoFish specification could lower to both a hypergraph and a spatial representation.

Having both structural and spatial information available suggests a different treatment of spatial movement than Benthic's parent context switching. For example, a spatial layer could offer directional and continuous information, such as the nearest meaningful element in a direction or the next element along a rope. The runtime could present additional spatial actions to extend the existing set of actions available to users at a node, instead of adding a separate spatial navigation mode. The design of this spatial layer remains future work.

Olli as an extensible platform for future research. Extensible domains let researchers and developers add accessible navigation for new kinds of graphic without reimplementing core features such as the navigation runtime. Structured navigation already exists for other kinds of graphic. TADA navigates node-link diagrams [14], and Wimer et al. argue for relational access to data structure diagrams [13, 12]. Each of these kinds of graphic could become a domain that supplies a lowerer and inherits Olli's runtime.

Olli also supports empirical studies of screen reader navigation. For example, Chundury et al. [2] used Olli as the screen reader condition in a comparative study of accessible data exploration systems alongside sonification and tactile displays. In the future, an Olli integration for reVISit [3], a framework for deploying visualization user studies, could let researchers run navigation studies on the shared runtime.

References

[1] M. Blanco, J. Zong, and A. Satyanarayan. Olli: An Extensible Visualization Library for Screen Reader Accessibility. In *IEEE VIS Posters*, vol. 6, p. 65, 2022. 1, 2

[2] P. Chundury, J. B. Jordan, Y. Reyazuddin, N. Elmqvist, and J. Lazar. Sound, Touch, or the Full Monty? A Comparative Study of Accessible Data Exploration Systems for Blind Users. *ACM Transactions on Accessible Computing*, 19(1):1:1–1:41, Mar. 2026. doi: 10.1145/3798100 4

[3] Z. Cutler, J. Wilburn, H. Shrestha, Y. Ding, B. Bollen, K. A. Nadib, T. He, A. McNutt, L. Harrison, and A. Lex. ReVISit 2: A Full Experiment Life Cycle User Study Framework. *IEEE Transactions on Visualization and Computer Graphics*, 32(1):13–23, Jan. 2026. doi: 10.1109/TVCG.2025.3633896 4

[4] F. Elavsky, L. Nadolskis, and D. Moritz. Data Navigator: An Accessibility-Centered Data Navigation Toolkit. *IEEE Transactions on Visualization & Computer Graphics*, 30(01):803–813, Jan. 2024. doi: 10.1109/TVCG.2023.3327393 1, 2, 3

[5] J. Gorniak, Y. Kim, D. Wei, and N. W. Kim. VizAbility: Enhancing Chart Accessibility with LLM-based Conversational Interaction. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, pp. 1–19. Association for Computing Machinery, New York, NY, USA, Oct. 2024. doi: 10.1145/3654777.3676414 2

[6] Highsoft AS. Highcharts, 2021. 2

[7] S. Jones, I. Pedraza Pineros, D. Hajas, J. Zong, and A. Satyanarayan. "Customization is Key": Reconfigurable Content Tokens for Accessible Data Visualizations. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, pp. 1–14. Association for Computing Machinery, New York, NY, USA, May 2024. doi: 10.1145/3613904.3641970 1, 2

[8] C. Mei, J. Pollock, D. Hajas, J. Zong, and A. Satyanarayan. Benthic: Perceptually Congruent Structures for Accessible Charts and Diagrams. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '25, pp. 1–17. Association for Computing Machinery, New York, NY, USA, Oct. 2025. doi: 10.1145/3663547.3746342 1, 2, 3, 4

[9] J. Pollock, C. Mei, G. Huang, E. Evans, D. Jackson, and A. Satyanarayan. Bluefish: Composing Diagrams with Declarative Relations. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, pp. 1–21. Association for Computing Machinery, New York, NY, USA, Oct. 2024. doi: 10.1145/3654777.3676465 1

[10] J. Pollock and A. Satyanarayan. GoFish: A Grammar of More Graphics! *IEEE Transactions on Visualization and Computer Graphics*, 32(1):549–559, Jan. 2026. doi: 10.1109/TVCG.2025.3634250 4

[11] J. R. Thompson, J. J. Martinez, A. Sarikaya, E. Cutrell, and B. Lee. Chart Reader: Accessible Visualization Experiences Designed with Screen Reader Users. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, pp. 1–18. Association for Computing Machinery, New York, NY, USA, Apr. 2023. doi: 10.1145/3544548.3581186 2

[12] B. Wimer, R. Kanchi, J. Mankoff, and R. Metoyer. Diagrams Are Structures: Reframing Accessibility for Data Structure Diagrams. In *Proceedings of the 2026 Conference on Research on Equity and Sustained Participation in Engineering, Computing, and Technology*, RESPECT 2026, pp. 1–6. Association for Computing Machinery, New York, NY, USA, June 2026. doi: 10.1145/3796496.3811750 4

[13] B. L. Wimer, R. Kanchi, K. Frierson, V. Potluri, R. A. Metoyer, J. Mankoff, M. Natsuhara, and M. X. Wang. Nonvisual Support for Understanding and Reasoning about Data Structures. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, pp. 1–21. Association for Computing Machinery, New York, NY, USA, Apr. 2026. doi: 10.1145/3772318.3791656 4

[14] Y. Zhao, M. A. Nacenta, M. A. Sukhai, and S. Somanath. TADA: Making Node-link Diagrams Accessible to Blind and Low-Vision People. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, pp. 1–20. Association for Computing Machinery, New York, NY, USA, May 2024. doi: 10.1145/3613904.3642222 4

[15] J. Zong, C. Lee, A. Lundgard, J. Jang, D. Hajas, and A. Satyanarayan. Rich Screen Reader Experiences for Accessible Data Visualization. *Computer Graphics Forum*, 41(3):15–27, Aug. 2022. doi: 10.1111/cgf.14519 1, 4

[16] J. Zong, I. Pedraza Pineros, M. K. Chen, D. Hajas, and A. Satyanarayan. Umwelt: Accessible Structured Editing of Multi-Modal Data Representations. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, pp. 1–20. Association for Computing Machinery, New York, NY, USA, May 2024. doi: 10.1145/3613904.3641996 1, 2, 3