Space$_X$Mag: An Automatic, Scalable, and Rapid Space Compactor for Optimizing Smartphone App Interfaces for Low-Vision Users

MD TOUHIDUL ISLAM, Pennsylvania State University, United States
SYED MASUM BILLAH, Pennsylvania State University, United States

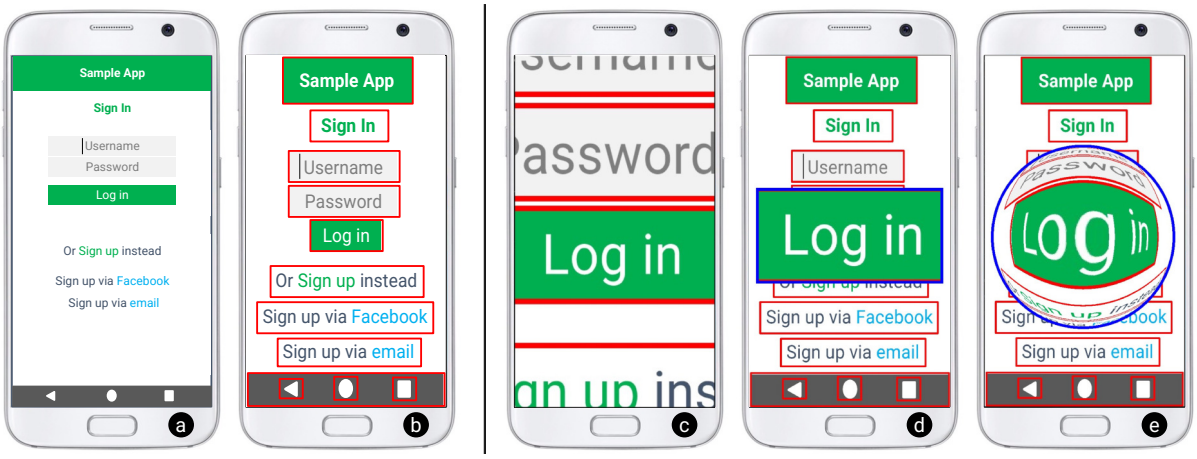


Fig. 1. Illustrations of Space$_X$Mag in three magnification modes. (a) shows the UI of an unmagnified app, and (b) shows its space-compacted UI produced by Space$_X$Mag. Notice that (b) is functionally equivalent to (a) but contains less whitespace; thus can pack more content in magnified form in the same screen real estate. Space$_X$Mag also applies a discernible border (e.g., red-colored border) to make UI elements easier for low-vision users to see and interact with. (c)-(e) show how the space-compacted UI appears in 3 magnification modes: (c) *fullscreen*; (d) *window*; and (e) *fisheye*.

Low-vision users interact with smartphones via screen magnifiers, which uniformly magnify raw screen pixels, including whitespace and user interface (UI) elements. Screen magnifiers thus occlude important contextual information, such as visual cues, from the user's viewport. This requires low-vision users to pan over the occluded portions and mentally reconstruct the context, which is cumbersome, tiring, and mentally demanding. Prior work aimed to address these usability issues with screen magnifiers by optimizing the representation of UI elements suitable for low-vision users or by magnifying whitespace and non-whitespace content (e.g., text, graphics, borders) differently. This paper combines both techniques and presents SpaceXMag, an optimization framework that automatically reduces whitespace within a smartphone app, thereby packing more information within the current magnification viewport. A study with 11 low-vision users indicates that, with a traditional screen magnifier, the space-optimized UI is more usable and saves at least 28.13% time for overview tasks and 42.89% time for

Authors' addresses: Md Touhidul Islam, Pennsylvania State University, University Park, PA, United States, mqi5127@psu.edu; Syed Masum Billah, Pennsylvania State University, University Park, PA, United States, sbillah@psu.edu.

2474-9567/2023/6-ART59 $15.00
https://doi.org/10.1145/3596253

target acquisition tasks, compared to the original, unoptimized UI of the same app. Furthermore, our framework is scalable, fast, and automatable. For example, on a public dataset containing 16,566 screenshots of different Android apps, it saves approximately 47.17% of the space (area) on average, with a mean runtime of around 1.44 seconds, without requiring any human input. All are indicative of the promise and potential of SpaceXMag for low-vision screen magnifier users.

CCS Concepts: • **Human-centered computing** → *Interaction paradigms*; **Accessibility technologies**; *Mobile devices*; **User interface design**.

Additional Key Words and Phrases: Low-vision Users, Screen Magnifiers, Zoom Lenses, Fisheye Effect, (Focus+Context)-based Magnification Techniques.

ACM Reference Format:
Md Touhidul Islam and Syed Masum Billah . 2023. Space$_X$Mag: An Automatic, Scalable, and Rapid Space Compactor for Optimizing Smartphone App Interfaces for Low-Vision Users. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 2, Article 59 (June 2023), 36 pages. https://doi.org/10.1145/3596253

1 INTRODUCTION

Low vision is a spectrum of vision impairments that cannot be fully corrected with glasses, medication, or surgery; individuals with low vision can lose the peripheral or central vision or have blurred vision, extreme light sensitivity, tunnel vision, and near-total blindness [11, 17]. This population relies on screen magnifiers, such as Zoom [3] in iPhone, Magnification in Android [1], to interact with smartphones. A typical screen magnifier uniformly magnifies screen content (e.g., 300% or 3×), including whitespace, as a blanket operation, causing the occlusion of important contextual information from the user's viewport. Consequently, low-vision users must manually pan over these occluded portions and mentally reconstruct the contextual information necessary for interaction [48]. This interaction is slow, cumbersome, and error-prone, as well as physically tiring and mentally demanding [22, 66, 74, 75].

Prior work has proposed two techniques, broadly categorized, to partly address these usability issues with uniform magnification. The **first** technique is *user interface (UI) adaptation* (e.g., [33, 35–38]) to cater to individuals' preferences and abilities. Researchers adapt the UI elements of an app for low-vision users by framing the magnification as an *optimization* problem, where the *constraints* include device types (desktop, mobile), screen sizes, input methods (touch-based, pointer-based), and users' preferences. The *cost function* equals the user effort required to interact with the magnified UI elements.

The **second** technique is *differential magnification* (e.g., [12, 21, 22]), where, instead of applying a uniform magnification throughout, different levels of magnification are applied to different UI elements. For example, SteeringWheel [22] applies less magnification to whitespace within a single UI or a UI group than the non-whitespace content (e.g., texts, graphics, borders) so that the non-whitespace content stays as close as possible. This preserves the context of UI elements within the current magnification viewport.

Both techniques have notable strengths and limitations. For instance, UI adaptations (via optimization) are more capable and can automatically find an optimal UI representation for low-vision users. In addition, this representation can eliminate the need for a separate screen magnifier. However, the current implementations are computationally expensive and require some human input (e.g., interaction traces, user preferences, and design pattern specifications); hence these are not fully automatic nor work for an arbitrary app. In contrast, differential magnification techniques are fast and run near real-time (i.e., less than 250 ms). However, their current implementations are ad hoc and domain-specific (e.g., demonstrated on webpages); require access to the *meta-representation* of an app, such as bounding rectangles, types, and parent-child relationships of all UI elements within the app; and the users must use a separate screen magnifier.

This paper presents **Space$_X$Mag**, combining the strengths of both techniques while minimizing their limitations. Space$_X$Mag automatically reduces whitespace in smartphone apps to facilitate differential magnification. It

frames space reduction as a *non-linear optimization* problem. The *constraints* include the alignment, hierarchy, and relative positions of UI elements before magnification; and the *cost function* equals the *total whitespace* in the screen. Space$_X$Mag takes the meta-representation of the UI elements in a mobile app and produces an optimally space-compacted meta-representation of the same elements. This meta representation can easily be extracted from the built-in accessibility supports on iOS [19] and Android [16] operating systems. Further, it does not need human input; thus, it is scalable and works for an arbitrary app in the wild. By design, Space$_X$Mag preserves the UI elements' spatial layout and relative positions in the original app, as low-vision users do not want a completely "different" app (or website) [76]. Furthermore, informed by prior work, Space$_X$Mag applies a discernible visual marker (e.g., red color) to indicate each UI element, the current focus, and the magnification viewport (if applicable) [22, 34, 48]. The before and after optimization of an app's UI is shown in Figure 1.a (before) and Figure 1.b (after).

We demonstrate Space$_X$Mag's effectiveness in three possible magnification modes [18, 26, 48, 69]: (i) *fullscreen mode*, when the magnification viewport takes up the entire screen (see Figure 1.c); (ii) *window mode*, when the viewport is a rectangular window (smaller than the screen size) that follows the user's cursor but occludes the unmagnified content (see Figure 1.d); and (iii) *fisheye mode*, when the viewport is circular that follows the user's cursor but does not occlude the unmagnified content (see Figure 1.e).

Our study with 11 low-vision participants showed that the participants took 28.13% less time (statistically significant) to gain an overview of interactive screenshots of different apps with a screen magnifier using space-optimized UI produced by Space$_X$Mag, compared to the original UI (baseline), in all three modes. Similarly, they took at least 42.89% less time to acquire targets using space-optimized UI than the baseline, which was also statistically significant. Furthermore, the participants reported that the space-optimized UI is more usable and makes them more aware of their current context (e.g., focus, screen location), as it packs more information within the magnification viewport. Our study also revealed that Space$_X$Mag is most effective in window mode over fullscreen or fisheye mode. We additionally discovered that UI elements' color, contrast, font, and layout are important contributors to participants' performance.

We also demonstrate the scalability of Space$_X$Mag. To that end, we ran it on a public dataset, RICO-SCA [7], containing 25,677 screenshots of different Android apps with *correct* meta-representations (e.g., no UI element has a negative coordinate or a bounding rectangle of area 0; and no UI element overlaps with another non-children UI element in its meta-representation). On average, Space$_X$Mag saved approximately 47.17% space (area) per screenshot, with a mean runtime of around 1.44 seconds. Compared to prior work on UI adaptation, such as SUPPLE [35], which can take 13 seconds to 6 minutes to optimize, Space$_X$Mag's runtime only takes 0.21 to 6.06 seconds (60× faster).

We summarize our contributions as follows:

- We proposed Space$_X$Mag, an optimization framework to automatically produce a space-compacted meta-representation of a smartphone app (§3.1, §3.3).
- We demonstrated Space$_X$Mag in two commonly used magnification modes (e.g., fullscreen and window) for low-vision users, as well as in fisheye mode, which is unavailable to current screen magnifiers (§4).
- We presented a study with 11 low-vision users to evaluate Space$_X$Mag in performing common magnification tasks (e.g., overview, target acquisition), understanding user experience, and discovering user strategies and different magnification modes (§6).
- We demonstrated Space$_X$Mag's scalability by running it on a public dataset, RICO-SCA [7], and reported metrics such as the average percentage of space saved and average runtime (§7).

2 BACKGROUND AND RELATED WORK

This section discusses earlier research on magnification types, dynamic adaptation of user interfaces, and optimization model types related to our research's design decisions.

2.1 Uniform and Non-uniform Magnification

Applications and webpages at their default scale (i.e., 1x or 100%) are often difficult to see for people with low vision [75]. A solution to this problem is screen magnification. There are two basic types of magnification: uniform and non-uniform. Uniform magnification involves applying one specific level of magnification (e.g., 150% or 200%) to each pixel on the screen. Popular magnifiers such as ZoomText [14], Windows Magnifier [9], and iOS Accessibility Zoom [3] make use of this kind of magnification. Consequently, low-vision users mostly use such magnification with basic zoom and pan.

From a technical point of view, uniform magnification is easily achievable by using affine transformations [4] and scale matrices [8]. However, such magnification limits the portion of the screen the user can see at once. To get to the other portions of the screen, users must use panning—either by using the mouse cursor (on desktops) or the touch and drag gesture (on touchscreen devices). Needing to pan is a well-known issue brought on by the uniform magnifier's limited viewport, as explored by Kline et al. [48].

Non-uniform magnification, on the other hand, works by applying different magnification levels on different objects on the screen. iOS Accessibility Zoom [3] offers a window magnification mode where only the contents inside a window get magnified (like a magnifying glass), and anything outside the window remains at their default magnification level. Other examples include fisheye lenses [18, 26, 69] and applying locality preserving magnification in a webpage [22]. Billah et al. [22] proposed identifying the semantically related local groups in webpages and preserving their connections before and after magnification. To achieve such a result, the authors used a space-reduction algorithm that modifies the size of the object boundaries in HTML. However, the authors only provided a pseudo-code for webpages, with no mention of an open-source library or package and no instructions on how to generalize the algorithm for non-HTML content. On the other hand, this paper provides an automatic space optimization framework that is scalable (§7), evaluated on smartphone UIs but easily extensible to desktop apps and webpages, and readily integrates into the graphics rendering pipeline of different operating systems (§8).

2.2 Bifocal Display Techniques

A type of non-uniform magnification is the bifocal display technique. We can categorize such techniques into two main types—i) **Overview and Detail** [46, 63], ii) **Focus and Context** [61]. In the former type, a small window (i.e., overview) on the screen (usually at the top-right or the bottom-right) shows the location of the viewport with regards to the entire screen [46, 63]. The overview window is only meaningful when some magnification (>100%) is already applied to the original screen. While this magnification mode is beneficial in some tasks such as basic UI navigation [45, 59, 62], its limitations appear in other tasks such as locating a small locality in a very densely populated map [61]. This method is not particularly helpful for low-vision people as the fraction of pixels that can be dedicated to the overview window is very small.

On the other hand, the latter type, i.e., the **(focus + context)**-based magnification techniques [41, 49, 61] is capable of displaying additional information in the context view (e.g., names, abbreviations, or distinct visual properties of the localities). Such a feature enables the user to get more contextual information and navigate easily. (Focus + context) techniques are also shown to be effective when users browse dense websites requiring to focus on small portions [49], perform large steering tasks [43], or attempt to select small targets with a stylus [67]. Agarwal et al. [13] designed *WidgetLens*, a (focus + context)-based magnification tool to work with tiny widgets on displays with high pixel density.

The apparent limitation of (focus + context) magnification techniques is associated with how the transition between focus and context is handled [28, 42]. Most techniques to achieve such transitions are inspired by real-world objects and phenomenons such as magnifying glasses, rubber sheets [70], and general surface deformations [27]. However, a primary magnifying glass lens creates occlusion of the content nearby to the focus [68]. Graphical fisheye lenses [69] solve the occlusion problem but introduces a problem of its own by distorting the contents in the transition space. Such distortion makes target acquisition at high magnification levels tiresome [42]. To reduce the impact of space distortion, researchers have explored the use of additional dimensions such as time [42] and translucence [61].

In summary, each non-uniform magnification technique has advantages and limitations. As such, we include magnifying glass (in window mode) and a fisheye lens (in fisheye mode) mode in our prototype design (§4).

2.3 User Interface Adaptation

2.3.1 Human Preference-based UI Adaptation. UI adaptation based on human (e.g., user, developer) choices and device characteristics has attracted some attention over the years [35, 37, 38, 53, 58, 64, 65], having its origins in the domain of automated design tools [73]. Cardelli [25] advocated decoupling the UI generation from the backend application program. The work included several tools to generate/manipulate essential UI components (e.g., cursor, text, pop-up dialog, window) until they met the designer's needs. Fogarty et al. proposed GADGET [33], an experimental toolkit that could generate optimization concept abstractions (e.g., decreasing UI item overlap, optimizing cluster shapes in grids), assisting designers in using optimization as a strategy to design UIs.

The most notable work in the domain would be SUPPLE as proposed by Gajos et al. [35–37]. The SUPPLE system considered UI rendering an optimization problem—with the device type (e.g., desktop computers, mobile devices), screen size, input method (e.g., touch-based, pointer-based), and users' preferences (e.g., layout) as constraints, and required user effort to use the UI as the cost function. Besides choosing the optimal layout, the SUPPLE system also picked the individual widgets to be rendered in the UI. The same authors later proposed SUPPLE++ [38], an extension of SUPPLE where the model also adapted according to users' vision and motor capabilities, in addition to its original functionalities.

Tools prior to SUPPLE like XIML [65], Pebbles project [58], and iCrafter [64] all lacked adaptability for supporting a wide range of devices and screen sizes. Lin et al. proposed Damask [53], a tool that can adapt UIs for other devices once it is designed for one. While designing for the first device or scenario using Damask, the designer has to specify the relevant design patterns for the UI. Despite offering such extensibility options, Damask failed to cope with situations where the device constraints are hard to anticipate [35].

A limitation of all these approaches is that they require some level of human input (i.e., interaction traces, user preferences, and design pattern specifications). On the other hand, what we are proposing adapts the UI for low-vision users without requiring external human input. However, we took inspiration from SUPPLE [35] and other prior works in similar trajectory [47, 58] to represent the UI item-specific information and inter-item relationships using a domain model (e.g., directed acyclic graph [5]).

2.3.2 Opportunistic UI Adaptation. Bigham [21] proposed automatic and gradual magnifications of a web UI as long as specific errors (e.g., horizontal scrolling, overlapping text, narrow word wrapping) do not appear. On average, this approach achieved 1.6x (i.e., 160%) magnification without introducing the aforementioned specific errors. However, the author did not claim this approach to be a robust solution for people with visual impairments such as low vision. A closely related work is Alotaibi et al.'s SALEM [15], a `Java` tool to automatically fix size-related accessibility issues (e.g., too small buttons or icons) in Android applications. However, this approach requires access to an app's `APK` file as well as an accessibility report and can take a substantial amount of time ($\approx$ 19 minutes in the worst case) to generate the output `APK`.

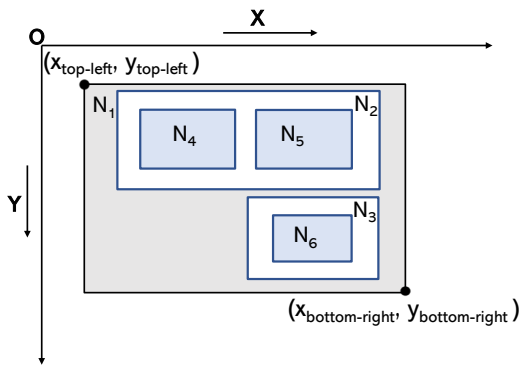


Fig. 2. A toy application with five UI elements in a coordinate system (origin: top-left corner). N_1 is the top-level UI containing N_2 and N_3; N_2 contains N_4 and N_5; and N_3 UI contains N_6. The bounding rectangles are shown in different colors. Y-axis grows from top to bottom.

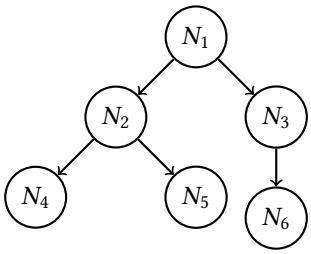


Fig. 3. The UI hierarchy of the top application shown in Figure 2. We represent this hierarchy as a directed acyclic graph (DAG), where each node represents a UI element, and each edge represents the parent-child relationship between two nodes (from the parent to the child).

Billah et al. [22] proposed a locality-preserving magnification technique ('SteeringWheel') in webpages—where the authors preserve the proximity of locally connected groups before and after the adaptation. The usefulness of this approach, however, was demonstrated via the use of a physical dial. Later, Lee et al. [51] proposed 'TableView', a tool that can extract website information, build relationships among UI items, and render them in a tabular layout, needing significantly less space than a typical website. The tool showed significant performance improvement over 'SteeringWheel' with low-vision users doing specific tasks (e.g., job search, shopping). The same authors later proposed 'TableView+' [50], which gained another minor improvement in task completion times over 'TableView'. Unfortunately, all of these approaches are for web UIs only, with their current implementations mostly being ad hoc.

2.3.3 Machine Learning-based UI Adaptation. Mezhoudi et al. [54] introduced a machine learning-based approach to UI adaptations. First, the approach asks users to rate their satisfaction levels on different layouts. As the user moves through different layouts, an automated system records their ratings and later uses them to train a prediction model. This approach, again, requires user input to adapt the UI—something our method does not need. Duan et al. [32] proposed a neural network model, to find a UI that has the minimum error rate and task completion times through the optimization process. The model used a manually-labeled dataset and had minimal attention to making contents bigger to make them accessible for low-vision users. Wu et al. [80] proposed Reflow, a technique where the UI layout and items are identified using a computer vision-based approach, an optimization (to minimize UI item selection times) is applied to the layout, and the app is re-rendered according to the new layout. The approach, however, contains no information on runtime, which is an important factor in UI adaptation to make the experience seamless.

3 SPACE$_X$MAG: PROBLEM FORMULATION

3.1 Mathematical Representation of User Interfaces

An app contains two types of UI elements: containers and leaves. The containers are parent elements (e.g., groups, subgroups, tables, frames, localities) that contain leaves. For example, N_1, N_2, and N_3 in Figure 2 are containers. Leaves are atomic elements (e.g., a button, an icon, a specific text), such as N_4, N_5, and N_6 in Figure 2. The bounding

rectangle of each UI can be represented by the coordinates of its two corners: top-left, ($x_{\texttt{top-left}}$, $y_{\texttt{top-left}}$); and bottom-right, ($x_{\texttt{bottom-right}}$, $y_{\texttt{bottom-right}}$). The parent-child relationships of UI elements can be captured by a Directed Acyclic Graph (DAG), $\mathbb{G} = \{\mathbb{N}, \mathbb{E}\}$, where $\mathbb{N}$ is the set of all UI elements (both types), and $\mathbb{E}$ is the set of all directed edges; the direction pointing from the parent to the child. Figure 3 shows that DAG for the UI elements in Figure 2.

3.1.1 Node Attributes. We assume each $N_i \epsilon \mathbb{N}$ has at least **four attributes**: $\texttt{Coord}(N_i)$, $\texttt{Width}(N_i)$, $\texttt{Height}(N_i)$, and $\texttt{Area}(N_i)$. $\texttt{Coord}(N_i)$ returns the bounding rectangle of N_i a tuple of four corners along X and Y axes (as shown in Figure 2); $\texttt{Width}(N_i)$, $\texttt{Height}(N_i)$, and $\texttt{Area}(N_i)$ return its width (along X-axis), height (along Y-axis) and area, respectively. Mathematically, these attributes are defined as follows:

$$\texttt{Coord}(N_i) = < x^{(\texttt{i})}_{\texttt{top-left}}, y^{(\texttt{i})}_{\texttt{top-left}}, x^{(\texttt{i})}_{\texttt{bottom-right}}, y^{(\texttt{i})}_{\texttt{bottom-right}} > \tag{1a}$$

$$\texttt{Width}(N_i) = \mid x^{(\texttt{i})}_{\texttt{bottom-right}} - x^{(\texttt{i})}_{\texttt{top-left}} \mid \tag{1b}$$

$$\texttt{Height}(N_i) = \mid y^{(\texttt{i})}_{\texttt{bottom-right}} - y^{(\texttt{i})}_{\texttt{top-left}} \mid \tag{1c}$$

$$\texttt{Area}(N_i) = \mid x^{(\texttt{i})}_{\texttt{bottom-right}} - x^{(\texttt{i})}_{\texttt{top-left}} \mid * \mid y^{(\texttt{i})}_{\texttt{bottom-right}} - y^{(\texttt{i})}_{\texttt{top-left}} \mid \tag{1d}$$

3.1.2 Binary Indicator Functions. We also define four indicator functions, $\delta_{\texttt{Left}}(.\ ,.)$, $\delta_{\texttt{Right}}(.\ ,.)$, $\delta_{\texttt{Above}}(.\ ,.)$, and $\delta_{\texttt{Below}}(.\ ,.)$ below, to test the relative position of any two nodes: N_i and N_j. For example, in Figure 2, N_4 is on the left of N_5. Therefore, $\delta_{\texttt{Left}}(N_4, N_5)$ will return 1 (*TRUE*), whereas $\delta_{\texttt{Right}}(N_4, N_5)$ will return 0 (*FALSE*). Conversely, $\delta_{\texttt{Left}}(N_5, N_5)$ will return 0, but $\delta_{\texttt{Right}}(N_5, N_4)$ will be 1.

$$\delta_{\texttt{Left}}(N_i, N_j) = \begin{cases} 1 & \text{if} \quad x^{(\texttt{i})}_{\texttt{bottom-right}} \leq x^{(\texttt{j})}_{\texttt{top-left}} \\ 0 & \text{otherwise} \end{cases} \tag{2a}$$

$$\delta_{\texttt{Right}}(N_i, N_j) = \begin{cases} 1 & \text{if} \quad x^{(\texttt{i})}_{\texttt{top-left}} \geq x^{(\texttt{j})}_{\texttt{bottom-right}} \\ 0 & \text{otherwise} \end{cases} \tag{2b}$$

$$\delta_{\texttt{Above}}(N_i, N_j) = \begin{cases} 1 & \text{if} \quad y^{(\texttt{i})}_{\texttt{bottom-right}} \leq y^{(\texttt{j})}_{\texttt{top-left}} \\ 0 & \text{otherwise} \end{cases} \tag{2c}$$

$$\delta_{\texttt{Below}}(N_i, N_j) = \begin{cases} 1 & \text{if} \quad y^{(\texttt{i})}_{\texttt{top-left}} \geq y^{(\texttt{j})}_{\texttt{bottom-right}} \\ 0 & \text{otherwise} \end{cases} \tag{2d}$$

3.1.3 Parent Indicator Function. Finally, we define a parent function, $\texttt{Parent}(N_i, N_j)$, and the cost function, $\texttt{Cost}(N_i)$. $\texttt{Parent}(N_i, N_j)$ returns 1 if node N_i is the parent of node N_j, or returns 0.

$$\texttt{Parent}(N_i,\ N_j) = \begin{cases} 0 & \text{if} \quad i = j \\ 1 & \text{if} \quad x^{(\texttt{i})}_{\texttt{top-left}} \leq x^{(\texttt{j})}_{\texttt{top-left}} \ \& \ y^{(\texttt{i})}_{\texttt{top-left}} \geq y^{(\texttt{j})}_{\texttt{top-left}} \ \& \\ & \quad x^{(\texttt{i})}_{\texttt{bottom-right}} \geq x^{(\texttt{j})}_{\texttt{bottom-right}} \ \& \ y^{(\texttt{i})}_{\texttt{bottom-right}} \leq y^{(\texttt{j})}_{\texttt{bottom-right}} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

3.1.4 Cost Function. $\texttt{Cost}(N_i)$ returns the cost for any node in the DAG. This function is used later in the UI optimization phase. The goal is to minimize the amount of whitespace inside a node. As such, the cost function is

defined as follows:

$$\texttt{Cost}(N_i) = \texttt{Area}(N_i) \; - \; \sum_{j=1}^{k} [\; \texttt{Area}(N_j) \mid j \neq i \;\; \& \;\; \texttt{Parent}(N_i, N_j) = 1 \;] \tag{4}$$

3.1.5 Defining an App as a Graph. With all of these attributes and functions defined, now we can formally represent our DAG, $\mathbb{G} = \{\mathbb{N}, \mathbb{E}\}$ as follows:

(1) $\mathbb{N} = \{N_1, N_2, N_3, ..., N_k\}$; considering there are k nodes in the DAG and
(2) $\mathbb{E} = \{\; (N_i, N_j) \;\mid\; i = [1, k],\; j = [1, k],\;\; \& \;\; \texttt{Parent}(N_i, N_j) = 1\}$.

3.2 Optimization Parameters and Constraints

3.2.1 Optimization Parameters. We optimize the bounding rectangles (four coordinates for each rectangle) of all the nodes in $\mathbb{G}$. Our optimization engine changes the values of these parameters (i.e., places the nodes in different places) following the predetermined constraints (described next) to find the combination that minimizes the whitespace. Thus, if $\mathbb{G}$ has n nodes, we need to optimize $4 * n$ parameters. Mathematically, we can write the set of parameters as follows:

$$\texttt{Params} = \{\texttt{Coord}(N_i) \;\mid\; i = [1, k]\} \tag{5}$$

Next, we describe a set of *constraints* (i.e., conditions) that the optimizer must respect.

3.2.2 Constraint: Relative Positions of the UI Objects. Users expect no significant changes regarding the layout of a website or an interface before and after modifying a UI [22, 76]. The UI items' colors and relative positions should stay the same to maintain a specific layout. For example, node N_3 is below N_2 in Figure 2. Hence, one should not reorganize the UI in a way that places N_3 above N_2. Moreover, the light blue colors of the nodes N_4, N_5, and N_6 should also remain the same after adaptation. We detect the relative positions of the items from their coordinates in the original UI, using Equations 2a, 2b, 2c, and 2d.

Note that for a pair of nodes, (N_i, N_j), one of Equations 2a or 2b and one of Equations 2c or 2d can return true at the same time. In other words, an item can be on the left/right and above/below another. In such cases, both restrictions are enforced. To summarize, if N_i and N_j are two nodes in an original UI, and $N_i^{'}$ and $N_j^{'}$ are their transformed versions in the optimized UI, then all of the following conditions must be satisfied:

$$\begin{aligned} \delta_{\texttt{Left}}(N_i, N_j) = \delta_{\texttt{Left}}(N_i^{'}, N_j^{'}), \quad \delta_{\texttt{Right}}(N_i, N_j) = \delta_{\texttt{Right}}(N_i^{'}, N_j^{'}) \\ \delta_{\texttt{Above}}(N_i, N_j) = \delta_{\texttt{Above}}(N_i^{'}, N_j^{'}), \quad \delta_{\texttt{Below}}(N_i, N_j) = \delta_{\texttt{Below}}(N_i^{'}, N_j^{'}) \end{aligned} \tag{6}$$

3.2.3 Constraint: Parent-Child relationships. The parent-child relationships must be satisfied as well when performing the optimization. For example, nodes N_4 and N_5 are children of node N_2 in Figure 2. After optimization, these parent-child relationships should not change. Thus, the set of edges ($\mathbb{E}$) in the DAG, which denotes all the parent-child relationships, should remain the same before and after optimization. Let us say that $\mathbb{E}$ and $\mathbb{E}^{'}$ denote the sets of edges in the DAGs for the original and optimized UIs, respectively.

$$\begin{aligned} \mathbb{E} = \mathbb{E}^{'} \;\; \texttt{i.e.,} \\ \forall i, j \;\; \texttt{if} \;\; \texttt{Parent}(N_i, N_j) = 1, \texttt{ then } \; \texttt{Parent}(N_i^{'}, N_j^{'}) = 1 \; \texttt{and} \\ \forall i, j \;\; \texttt{if} \;\; \texttt{Parent}(N_i, N_j) \neq 1, \texttt{ then } \; \texttt{Parent}(N_i^{'}, N_j^{'}) \neq 1 \end{aligned} \tag{7}$$

Here N_i and N_j are two nodes in the original UI, and $N_i^{'}$ and $N_j^{'}$ are the same two nodes in the optimal UI.

3.2.4 Constraint: Not Resizing Children When Optimizing a Parent. Space$_X$Mag's optimization works in a bottom-up fashion for the UI hierarchy (§3.3.3). Hence, when optimizing a parent node, its children should already be in the optimal state. As such, the children will not be resized—they will only be moved around to see if their displacement can result in better space optimizations for the parent. In summary, when optimizing a node N_i,

$$\begin{aligned} \forall N_j \quad \texttt{if} \;\; \texttt{Parent}(N_i, N_j) = 1, \; \texttt{then}, \\ \texttt{Width}(N_j) = \texttt{Width}(N_j^{'}) \;\; \texttt{and} \\ \texttt{Height}(N_j) = \texttt{Height}(N_j^{'}) \end{aligned} \tag{8}$$

3.2.5 Other Constraints. Other constraints include retaining the original UI alignment (if any), not allowing a child to go outside its parent's bounding rectangle, ensuring that coordinates do not assume negative values, and requiring a parent node's area to be greater than or equal to the sum of its children's areas, among others.

3.3 Space$_X$Mag's Implementation

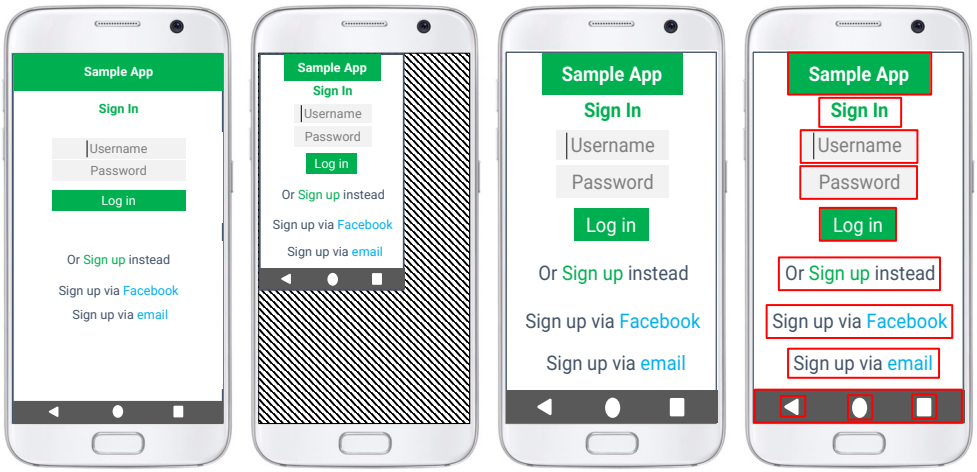
3.3.1 Optimization Engine. Since our cost function (Eq 4) represents a 2D quantity, the total area of whitespace, we used a non-linear optimization engine, GEKKO module [6]. This engine is built on Interior Point (IPOPT) [78] solver and specializes in dynamic optimization of non-linear, mixed-integer problems [20]. Moreover, GEKKO's implementation allowed us to easily incorporate the parameters (as *Variables*), constraints (as *Equations*), and cost function (as *Minimize* or *Maximize* method) into the model.

3.3.2 Input and Output. The primary input and output of our current implementation is the meta-representation of UI elements (i.e., View Tree); we take the original meta-representation and produce its optimized version. For demonstration, we also take a screenshot as the second input and output its space-optimized version. We sample input meta-representation and screenshots from the RICO-SCA dataset [7]. Note that our implementation is a reasonable approximation of the real-world scenario; if Space$_X$Mag is implemented in the Operating System (OS), we can update the View Tree and send it back to the View System, which then displays the space-compacted content on screen via the rendering pipeline. We elaborate more in Section 8.1.3.

3.3.3 Optimization Process. We apply a bottom-up optimization in the given UI hierarchy. First, we optimize each node before getting to its parent. We consider the DAG a tree to identify the node traversal order and apply a *post-order* traversal. For example, the traversal order for the UI in Figures 2 and 3 would be $\{N_4, N_5, N2, N_6, N_3, N_1\}$. Note that the orders $\{N_6, N_3, N_4, N_5, N2, N_1\}$ and $\{N_4, N_5, N_6, N_2, N_3, N_1\}$ are also valid. GEKKO engine then optimizes each node based on its parameters and constraints (described in Equation 5 and Section 3.2). Appendix A provides a step-by-step demonstration of how Space$_X$Mag works with a toy applicatoin.

3.3.4 Generating Output Images. Since we generate an image output for demonstration, we match the background color of the output image with the input screenshot. We detect the background color in the input image by finding the dominant color. Then, we create a blank image with this background. We also extract the individual UI items (i.e., leaves) using the original meta-representation. Once all optimization is complete, we render the previously extracted UI items according to the optimized meta-representation. Finally, we resize the optimized UI back to its original dimension. This process is shown in Figure 4(b)-(c). Compared to the original Figure 4(a), the optimized Figure 4(c) is magnified $\approx 1.6x$ by default. We update the output meta-representation accordingly to reflect this resizing.

3.3.5 Border Generation and Output. From our prior experience working with low-vision users, we understood that a rectangular border (preferably of bright color) drawn around a target UI object makes it easier for them to locate it. We used the `red` color for this demonstration to draw these borders. In addition, the widths of these



(a) Original UI (b) Space-optimized UI (c) Space-optimized UI (resized) (d) Space-optimized UI (with borders)

Fig. 4. Resizing and border generation phases during Space$_X$Mag's optimization.

borders depend on the dimension of the object it is being drawn around (i.e., a larger object would have thicker borders). The colors and widths of these borders are customizable. An optimal, resized, and object-bordered UI is shown in Figure 4(d). Figures 4(a)-(d) demonstrates how a UI looks before and after Space$_X$Mag optimizes its meta-representation.

4 CONSUMING SPACE$_X$MAG: A SIMPLE SCREEN MAGNIFIER

We also implemented a simple screen magnifier that allows low-vision users to zoom and pan a screenshot image (original or optimized) in three modes: fullscreen, window, and fisheye.

4.1 Fullscreen Mode

The fullscreen mode reflects the traditional pan-and-zoom approach utilized by magnification tools, such as ZoomText [14]. This mode is supported by most magnification tools available across different Operating Systems, making it the most widely used mode among low-vision users. The primary advantages of the fullscreen mode are convenience and popularity. The main disadvantage is its uniform magnification technique—where less helpful content, such as whitespace, is magnified just like more valuable content, such as buttons and texts. This often results in a loss of context and requires extensive panning.

4.2 Window Mode

Window mode works like a magnifying glass (Figure 5a). The name is inspired by the rectangular-shaped viewport it uses. The contents inside the viewport are magnified uniformly, and anything outside is left as-is. This mode boasts all the advantages of a typical `(focus + context)`-based magnification technique. It allows users to be aware of the context (i.e., content outside the viewport). However, this mode can cause occlusion of the contents close to the viewport when the applied magnification scale is high.

(a) Window Mode (b) Fisheye (Circular) (c) Fisheye (Oval) (d) Fisheye (Rect.)

Fig. 5. Outputs when zooming into a specific portion of a UI using window mode (a), and fisheye mode using different lens shapes (b-d). Notice that the distorted area in the rectangular (d) and oval (c) lens shapes are less than the circular (b) one.

4.3 Fisheye Mode

Fisheye mode is another example of `(focus + context)`-based magnification. The mode applies a non-uniform magnification to the contents inside the lens (i.e., the circular viewport), magnifying contents closer to the lens center and distorting contents that are further away. The level of magnification/distortion is controlled using a distortion function. Among many implementations of this distortion function, we used the *Sarkar-Brown* method [69]. We believed allowing low-vision users to change the level of distortion during a task could be distracting (and maybe tiring) to their eyes. As such, we decided to change the lens size whenever the user changed the magnification level using the two mouse buttons (see Table 1). This action made ensures that the distortion remained consistent throughout. Although a circle is the most common Fisheye lense, we implemented two additional lens shapes: an oval, as shown in Figure 5(c); and a rectangle, as shown in Figure 5(d). Compared to the circular lens, the latter two distort less area.

4.4 Supported Gestures and Shortcuts

Our prototype screen magnifier supports limited keyboard commands and mouse interaction (Table 1). These include zooming in/out, panning, and changing the size of the magnification viewport.

5 EVALUATION OF SPACE$_X$MAG

To understand the usability of Space$_X$Mag-optimized UIs in specific tasks (e.g., generating the overview of an image, locating a target) alongside magnification tools, we conducted an IRB-approved user study.

5.1 Participants

We recruited 11 low-vision participants (six males and five females) through participants' mailing lists, university mailing lists, and posts on public forums. The average age of the participants was 36.18 (min 21, max 59, SD 10.65). Our inclusion criteria were users with low vision who used screen magnification tools (e.g., ZoomText [14],

Table 1. Mouse/Keyboard mappings of different actions in the prototype.

Mouse/Keyboard Input	Function		
	Fullscreen Mode	Window Mode	Fisheye Mode
Left Mouse Buttom	Zoom In	Zoom In	Zoom In and Increase Lens Size
Right Mouse Button	Zoom Out	Zoom Out	Zoom Out and Decrease Lens Size
Mouse Movement	Pan	Pan	Pan
Space button	-	Change Window Size	Change Lens Shape

Table 2. Participants' demographics, medical condition, preferred magnifiers, and self-reported preferred magnification levels. FOV: Field of vision, ZT: ZoomText, ZT-F: ZoomText Fusion, WM: Windows Magnifier, i-ZM: iOS Zoom, M-ZM: MacOS Zoom, A-ZM: Android Zoom.

ID	Age/ Sex	Medical Condition	Visual Acuity	Profession	Magnifiers	Magnification
P1	33/F	Ocular Albinism, Photophobia	L: 20/200 R: 20/200	Non-Profit Association	WM, ZT, ZT-F	300%-350%
P2	29/F	Pseudotumor Cerebri	Unknown	Unemployed	WM	600%-700%
P3	59/M	Leber Hereditary Optic Neuropathy	L: 20/500 R: 20/500	Retired	WM, ZT, ZT-F	400%-450%
P4	35/F	Idiopathic Intracranial Hypertension	L: 20/200 R: Unknown	Employed (on Medical Disability)	WM, ZT	200%-300%
P5	37/F	Underdeveloped Optic Nerve	L: 20/400 R: 20/400	Independent living Instructor for senior	ZT-F, i-ZM	250%-350%
P6	35/F	Diabetic Retinopathy	L:20/200; R: 20/200	Data Dispatcher	ZT, i-ZM	150%-200%
P7	33/M	Pseudotumor Cerebri	Unknown	Professor (Ph.D.)	ZT, i-ZM	600%-700%
P8	38/M	Aphakia	Both: 5/400, 10° FOV	Digital Marketing	WM, ZT, A-ZM	250%-300%
P9	27/M	Pathological Myopia	Unknown	Graduate Student	WM, i-ZM	200%
P10	51/M	Congenital Cataracts	L: 20/200 R: 20/400	Occupational Therapy Assistant	ZT, WM, M-ZM	300%-400%
P11	21/M	Pathological Myopia	Unknown	Unemployed	WM, i-ZM	175%-200%

iOS Accessibility Zoom [3]). The details (e.g., eye condition, profession, preferred magnifiers) regarding the study participants are available in Table 2. All participants lived in the eastern states of the USA (e.g., Virginia, Pennsylvania, and New York).

5.2 Study Design

We used a within-subject design—all of our participants performed the following two tasks:

5.2.1 Task 1 (T1) : Overview Task. Since using a smartphone typically involves looking at the screen to interpret its contents, this task asked participants to look at a given screen and report its layout, possible purpose, and the objects of interest. Objects of interest included text fields, generic buttons, icons, click-on options, and scroll buttons, among others. For instance, in Figure 6(a), participants could report that it had a list-like layout. Similarly, in Figure 6(b), they could report that it was a `Login` page with 7 objects: `{'LOGIN section header', 'E-mail text field', 'Password text field', 'NEW HERE? section header', 'SIGN UP WITH option', 'E-MAIL button',`

`'FACEBOOK button'`}. Completion time for a task was later normalized using the number of objects a participant managed to report.

5.2.2 Task 2 (T2) : Target Locating Task. This task asked the participant to find a specific target UI (e.g., a button, a heading, an icon) on the screen. We provided textual and graphical (if any) descriptions of the target. One example would be asking a participant to read out the number in the captcha (e.g., 94101) in Figure 5(a). We also normalized the completion times in T2 using the number of total objects of interest in the UI.

Table 3. An overview of different study conditions for each task.

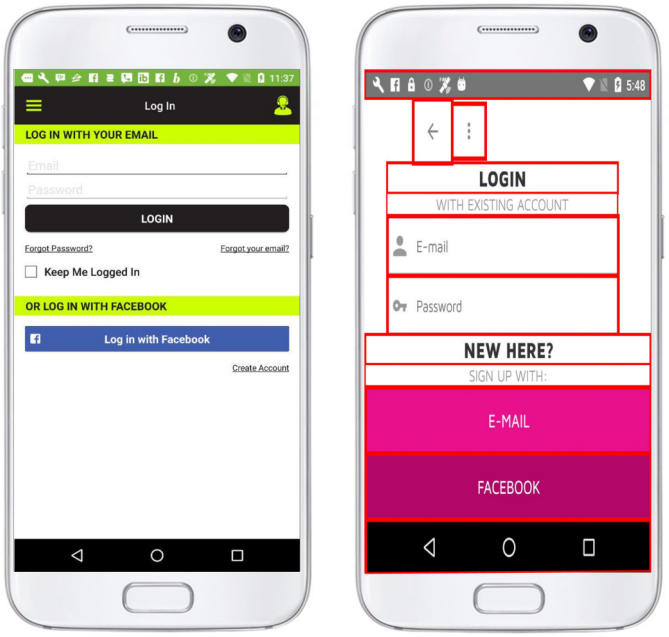
Serial	Screen Type	Magnification Mode	Study Condition
1	Original	Fullscreen	$C_{0.0}$
2	Space$_X$Mag's output	Fullscreen	$C_{1.0}$
3	Original	Window	$C_{0.1}$
4	Space$_X$Mag's output	Window	$C_{1.1}$
5	Original	Fisheye	$C_{0.2}$
6	Space$_X$Mag's output	Fisheye	$C_{1.2}$

5.2.3 Study Conditions and Task Block. With two types of screens (original, Space$_X$Mag's output) and three magnification modes (fullscreen, window, fisheye), we have six study conditions in total, as shown in Table 3. We assign a code for each study condition in the form of $C_{X.Y}$, where X denotes the screen type, and Y denotes the magnification mode. $X = 0$ means original screen (unoptimized UI) and $X = 1$ means Space$_X$Mag's output screen (optimized UI). On the other hand, $Y = 0$ means Fullscreen magnification mode, $Y = 1$ means Window mode, and $Y = 2$ means Fisheye mode. While comparing the original UIs and Space$_X$Mag-optimized UIs is the main focus of the study, we also wished to observe how the choice of magnification modes impacts the performance of our participants. Each task had three trials. In total, we recorded $6 * 2 * 3 = 36$ data points for each participant. The task ordering and the study conditions were counterbalanced among participants.

5.2.4 Choice of UIs for Different Tasks. We chose Android screens (i.e., images) from the RICO-SCA [7] dataset to conduct our study. We picked the unmodified RICO-SCA images for representing the original UIs (i.e., study conditions $C_{0.0}$, $C_{0.1}$, and $C_{0.2}$). On the other hand, we picked the outputs of Space$_X$Mag (input images also came from the RICO-SCA dataset) for representing study conditions $C_{1.0}$, $C_{1.1}$, $C_{1.2}$. To maintain consistency, we imposed some constraints while choosing UIs (i.e., images) for different tasks. For a specific magnification mode (one of fullscreen, window, or fisheye) and task (either T1 or T2), the two chosen images (one for original UI, the other for optimized UI) must have a similar layout (e,g, list, grid). We focused primarily on `List` and `Grid` layouts because i) they are the two major categories supported by *Android* layout design [40], and ii) the RICO-SCA (or, RICO in general) dataset consists of images (screenshots) from *Android* smartphones. Further, both screens should have the same or very close number of objects of interest (at most a difference of 3) in the UI. Figure 6 shows two sample screens used in the study.

5.3 Study Procedure

5.3.1 Setup. We conducted the study remotely using Zoom telecommunication software [10]. The feasibility of an in-person study was questionable, given the sudden uprise of COVID-19 during that time. All the necessary programs, e.g., the image viewer, Google Chrome browser (for NASA-TLX questionnaire), and Space$_X$Mag itself, ran on our test laptop computer, on Ubuntu 22.04.1. The computer was a Dell XPS 9700 with an 8-core CPU (Core i7 10875H) and 32 Gigabytes of RAM. The laptop had a 17-inch screen with a resolution of 1920 * 1200.



(a) A UI for study condition $C_{0.0}$ (b) A UI for study condition $C_{1.0}$

Fig. 6. Two sample screenshots used in the study: an original UI (left) and Space$_X$Mag's output (right). Both screens have a similar layout (`List`) and have a comparable number of objects of interest in the UI (9 on the left, 7 on the right).

We gave the participants remote control access (via Zoom [10]) to make sure that they can pan, zoom, and move the cursor as they feel necessary in our test machine. We asked the participants to turn off any local magnifiers (e.g., ZoomText, MacOS Accessibility Zoom) they could have been using. Two researchers were present in each study session. While one gave necessary instructions to the participant and conducted the study, the other took notes and timings for different tasks.

5.3.2 Questionnaires and Familiarizing with Space$_X$Mag. We started each study session by asking the participant demographic questions (e.g., age group, occupation, medical condition of the eye, preferred magnification tools, preferred zoom level). Afterward, we asked about their accommodation preferences, such as light/dark modes, colored texts, and font faces. We continued by asking about their experience with different magnification modes (e.g., fullscreen, window). Then, we described the fisheye lens, as it was unfamiliar to most participants.

After this initial QA session, we showed the participant two UIs side-by-side (as shown in Figure 8). On the left was an original UI, while on the right, we showed the output of Space$_X$Mag for the exact UI. We asked the participant for their reaction to the two UIs. In addition to space reduction, we also asked about the red borders' role (if any) in improving the visibility of UI content.

Next, we introduced the participant to our screen magnifier prototype. We demonstrated how they could use the three magnification modes. Then, the participant played around with different magnification modes (i.e., fullscreen, window, fisheye) and their operations (e.g., panning, zooming in and out, changing lens shape and size). When the participant felt comfortable using the prototype, we moved on to the next section of the study.

5.3.3 Procedure in T1. First, we described what a participant has to achieve in task T1 (§5.2.1). Apart from identifying the objects of interest, we also asked the participant to report the layout and probable purpose of the page. All of these factors were considered when normalizing (§5.5) the completion times during data analysis. We also allowed the participant to move on from a trial if they felt it was too hard for them. In such a case, we marked the trial as incomplete and excluded it from the analysis. If the participant needed help remembering the button/functions for a particular mode of magnification, we provided instant help.

5.3.4 Procedure in T2. For the target finding task (T2), we described that the participant had to find a specific target on the screen. We provided an extensive description for the target—textual and graphical, if available. We asked the participant to stop moving their cursor and give verbal confirmation to us when they felt they had found the target. If the participant was correct, we marked the task as complete and took down the timings. If the participant was wrong, we requested the participant to keep looking.

The difficulty in locating a target in a UI with hundreds of potential targets, when compared to a UI with only a few, is considerably different, both intuitively and from our experience in prior studies. As such, we also normalized the completion times in T2 using the number of objects of interest in a UI.

5.3.5 Mid-study and Post-study Questionnaires. After the participant was done with the first task (either T1 or T2, due to counterbalancing), we asked them about their overall experience, challenges, and recommendations. We also asked them to rate the three magnification modes according to their preference. We concluded the mid-study questionnaire by asking if they had observed any notable difference between the original UI and Space$_X$Mag's outputs. After the participant finished the second task, we asked them to rate the three modes again. In case they had a different order of preference before, and after the second task, we asked for possible explanations.

5.3.6 SUS and NASA-TLX evaluations. After a participant completed the 36 trials of T1 and T2, we asked them to rate the usability of Space$_X$Mag's output UIs using different magnification modes. We performed the first evaluation using the SUS questionnaire [23]—a set of ten Likert scale statements. For each statement, the participant rated their agreement/disagreement on a scale of 1-5, with 1 meaning `strongly disagree` and 5 meaning `strongly agree`. We also calculated the NASA-TLX load indices [44] for Space$_X$Mag's output UIs' usability for three magnification modes.

Each study session lasted around 2 hours, and we allowed the participant to take breaks whenever needed. We compensated each participant with a 50$ Amazon gift card.

5.4 Data Collection and Analysis

As the participants performed the tasks specified by the researcher conducting the study, their actions were recorded via Zoom's video recording and an internal logger to enable future data analysis. Our primary goal was to discover if there is any significant difference in participants' performance when working in original and Space$_X$Mag's output UIs.

We also analyzed the participants' data by creating different UI characteristic-based (e.g., layout, action, target type) subdivisions. Moreover, we followed recommendations for qualitative interview design [77]. Figure 7 shows examples of different UI layouts and prominent actions within the UI. We classified a UI as a `Grid` layout if the items were distributed in more than one column (e.g., Figure 7c-d). Otherwise, it was marked as a `List` layout (e.g., Figure 7a-b).

On the other hand, if a UI was predominantly comprised of text inputs, it was marked as a `Write`-prominent UI (e.g., Figure 7b, 7d). Otherwise, it was considered a `Read`-prominent UI (e.g., Figure 7a, 7c). Note that these classifications were not known to the participants but were created by us after the study in the data analysis phase. Each participant performed exactly 36 tasks as discussed in Section 5.2.3.

(a) List layout, Read Prominent UI

(b) List layout, Write Prominent UI

(c) Grid layout, Read Prominent UI

(d) Grid layout, Write Prominent UI

Fig. 7. Examples showing different UI layouts (list/grid) and prominent UI actions (read/write). Notice that a UI can have two different classifications at the same time, i.e., it can have a 'grid' layout and 'write' as its prominent action (d). Moreover, a target such as *"the flag of Spain"* (c) is considered of type *text+graphics*, while a target like *"Select language"* option (c) is considered of type *text*.

5.5 Data Normalization

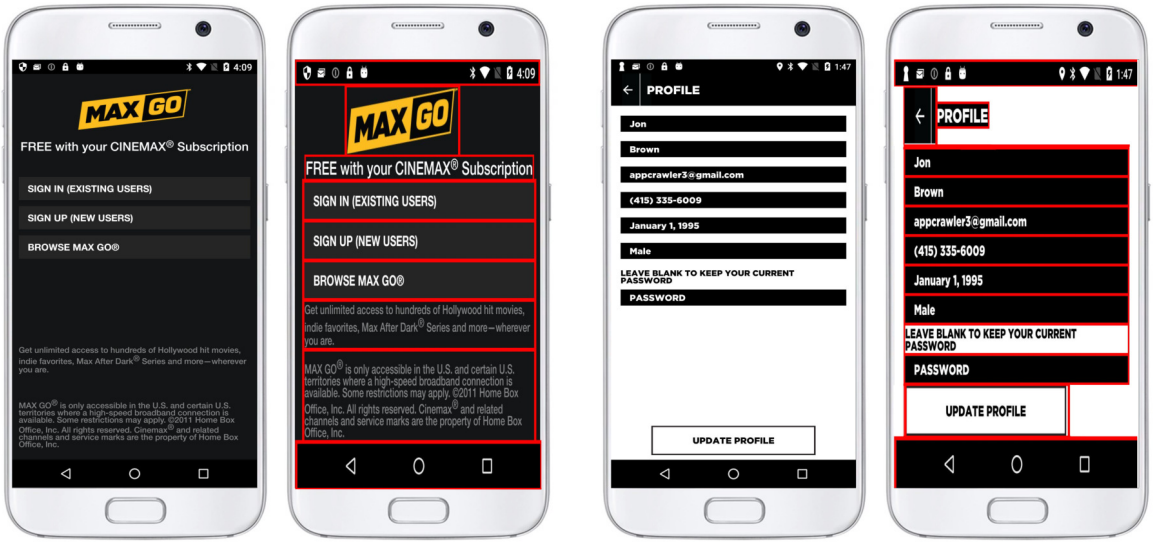
We normalized the completion times in T1 by dividing the time by the total number of objects of interest (§5.2.1) a participant managed to report. In T2, we divided the time by the number of the real object of interest in a UI. We found the completion times in our study to be not normally distributed via Shapiro-Wilk tests. As such, we ran a Wilcoxon Signed-Rank test when comparing two groups (e.g., completion times in original vs. Space$_X$Mag's output UIs) and a Kruskal-Wallis test when comparing more than two (e.g., completion times using three magnification modes).

6 FINDINGS

In this section, we first present a quantitative comparison of Space$_X$Mag's output and original UIs, including study participants' task completion times, SUS, and NASA-TLX scores. Then, we discuss their strategies and preferences when performing the study tasks (i.e., T1 and T2). Finally, we point out the additional magnification features they most commonly asked for.

6.1 Comparison of Space$_X$Mag's Output with Original UIs

6.1.1 Participants' Opinions On Space$_X$Mag's Outputs. When presented with the side-by-side view of an original UI (e.g., Figure 8a) and Space$_X$Mag output of the same UI (e.g., Figure 8b), all of our participants preferred the UI in Figure 8b. In addition, all the participants could notice the larger UI contents in Figure 8b. 4 (P4, P5, P9, P11) of our 11 participants found Figure 8b's contents large enough to read/understand without using any magnification tool. In the words of P5:



(a) (b) (c) (d)

Fig. 8. Original android screens from the RICO-SCA dataset (a and c) and Space$_X$Mag's output screens (b and d).

"*I feel like I don't have to zoom as much for the one on the right.*"

The general tone regarding Space$_X$Mag-generated UIs was predominantly positive. In the words of P1:

"*The one on the right (Space$_X$Mag's output) is exactly what a mobile app should look like.* "

6.1.2 Task Completion Times for T1 and T2. Normalized completion times in tasks T1 and T2 are shown in Figure 9a and 9b, respectively, for all six conditions as discussed in Table 3.

We subdivided the results based on the magnification mode used for a task. For both tasks, the normalized completion times in each magnification mode improved significantly from an original UI to Space$_X$Mag's output UIs.

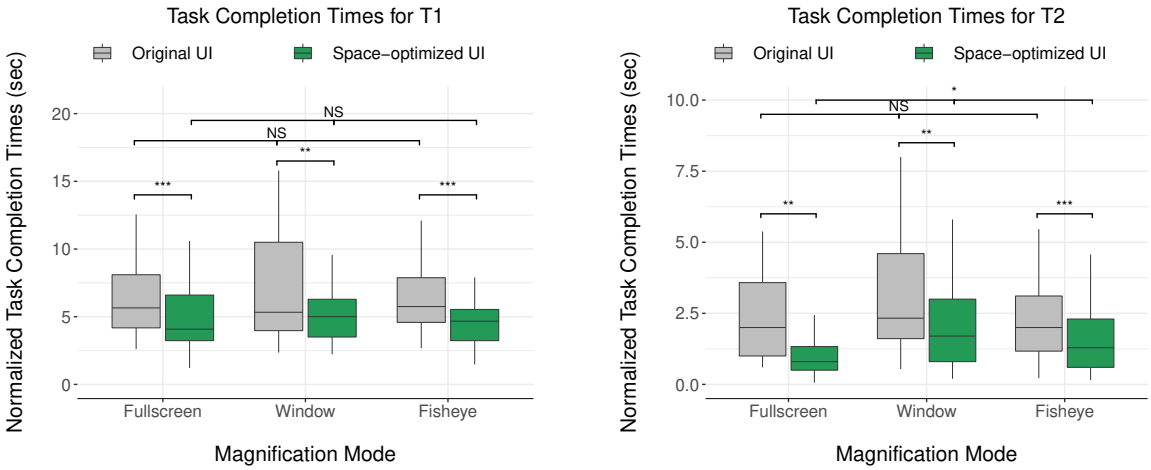
We also compared the normalized completion times among the three magnification modes. Apart from Space$_X$Mag's output UIs in T2, we found no statistically significant difference when using the three magnification modes. The summary of the statistical tests is shown in Table 4.

Combining all the magnification modes, we see a 28.13% and 42.89% decrease in normalized completion times for T1 and T2, respectively, when using Space$_X$Mag's output UIs compared to original UIs. In summary, while the magnification mode may not always impact the completion time, the type of UI used (e.g., original, Space$_X$Mag's output) does.

6.1.3 Task Completion times in Different UI Layouts. Figures 10a and 10b show the normalized task completion times in T1 and T2, respectively, with the UIs separated based on their layouts (e.g., list, grid).

For Figure 10a, the performance difference of original UIs in the two layouts is not substantial. However, this difference is very noticeable with the optimized UIs. We believe this happened because of the red borders drawn in the optimized UIs. According to P9 and P11, the borders in grid layouts are more helpful than in the list layouts.

We also notice a more substantial performance improvement between the two types of UIs in the grid layout. This is because the grid layout usually has a well-defined alignment of rows and columns. As such, this is



(a) T1 (b) T2

Fig. 9. Box-plots showing normalized task completion times for participants in T1 and T2 (lower is better). ***, **, and * represents a statistically significant difference with p<0.001, p<0.01, and p<0.05 respectively; while NS means no statistically significant difference. Detailed statistical test results are available in Table 4.

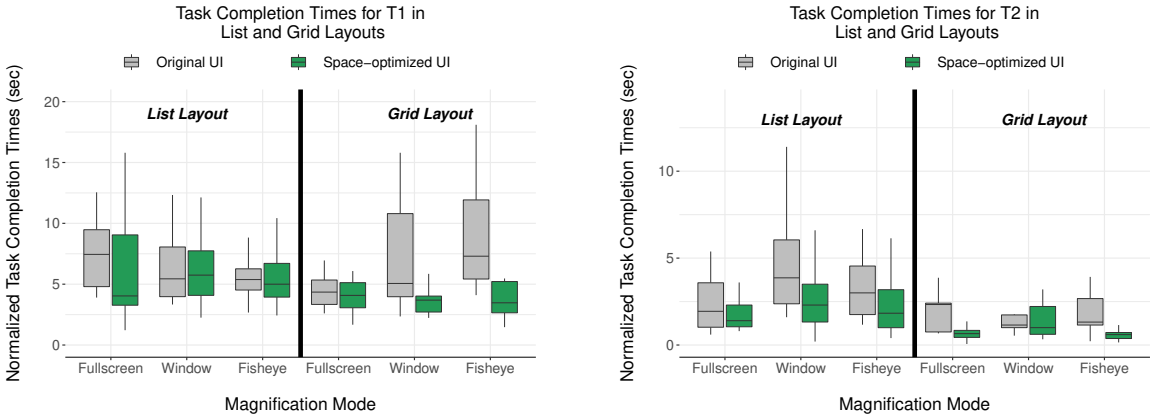
Table 4. Summary of the statistical tests in T1 and T2. ***, **, and * represents a statistically significant difference with p<0.001, p<0.01, and p<0.05 respectively; while NS means no statistically significant difference. The Wilcoxon Signed-Rank tests were two-tailed and the Kruskal-Wallis tests were right-tailed. α was 0.05 in all cases.

Comparison Study Conditions	Test Used	Result: (T1)	Result: (T2)
Original vs. Space-optimized UI (Fullscreen Mode)	Wilcoxon Signed-Rank	* * * (p = 0.00069, Z = 3.40)	** (p = 0.000015, Z = 4.32)
Original vs. Space-optimized UI (Window Mode)	Wilcoxon Signed-Rank	** (p = 0.0034, Z = 2.93)	** (p = 0.0021, Z = 3.077)
Original vs. Space-optimized UI (Fisheye Mode)	Wilcoxon Signed-Rank	* * * (p = 0.000024, Z = 4.23)	* * * (p = 0.0017, Z = 3.14)
Fullscreen vs. Window vs. Fisheye Mode (Original UI)	Kruskal-Wallis	NS (χ^2 = 0.016092, p = 0.992)	NS (χ^2 = 3.946, p = 0.139)
Fullscreen vs. Window vs. Fisheye Mode (Space-optimized UI)	Kruskal-Wallis	NS (χ^2 = 0.4165, p = 0.812)	* (χ^2 = 9.0159, p = 0.01102)

maintained after the optimization. List layouts, on the other hand, lack well-defined alignments. Thus, the improvements in this layout going from original UIs to optimized UIs are minor.

For Figure 10b (i.e., T2), most findings of T1 remain true. In addition, we see a performance difference in the list layout with original and optimized UIs.

6.1.4 Task Completion Times Based on Prominent UI Action. Figures 11a and 11b show the normalized task completion times in T1 and T2, respectively, with the UIs separated by their prominent action (e.g., read, write).



(a) T1 (b) T2

Fig. 10. Normalized task completion times in T1 (left) and T2 (right) based on the UI layout (e.g., list and grid). Lower is better. Layout classification process is discussed in Section 5.4.

(a) T1 (b) T2

Fig. 11. Normalized task completion times in T1 (left) and T2 (right) based on the prominent UI operation (e.g., read and write). Lower is better. Prominent action classification process is discussed in Section 5.4.

In both figures, we see drastic improvements from original UIs to optimized UIs when working on 'write' action-based UIs. Generally speaking, 'write' action-based UIs usually have more space (examples are Figures 6a, 6b). As such, Space$_X$Mag has more room to optimize. On the other hand, read action-based UIs are usually packed with contents, leaving little whitespace in between (examples are Figures 8c, 8d). Thus, Space$_X$Mag cannot improve the performance by a similar margin.

6.1.5 SUS Scores. Using the SUS questionnaire [71], we gathered feedback from our participants regarding the usability of Space$_X$Mag's output UIs with different magnification modes. The average score was 84.55, with an SD of 7.57. While we did not measure the usability of a baseline system (i.e., original UI with fullscreen magnification), we can get those data from previous studies. Momotaz et al. [55] reported the SUS score of such a

baseline magnifier to be 50.15 (SD 23.42). As such, Space$_X$Mag outperforms the baseline by a significant margin. Our SUS score of 84.55 has a percentile of more than 96 and attains the adjective "`Best Imaginable`" according to SUS scale interpretations [71]. Our participants predominantly hailed the usability of Space$_X$Mag. P3 and P4 appreciated the basic controls and convenient design of the prototype. P1 expressed a wish encompassing all magnifiers:

> "*I wish all magnifiers had this many options (of different magnification modes and lens shapes).*"

P3 was also very excited about the possible public deployment of Space$_X$Mag and believed it would new dimensions to how LV people use different UIs.

6.1.6 NASA-TLX Load Indices. Figure 15 shows the NASA-TLX load indices when using Space$_X$Mag's outputs with different magnification modes. Fullscreen (mean 62.26, SD 11.46), window (mean 42.18, SD 11.97), and fisheye (mean 58.55, SD 25.44) modes scored lower (i.e., better) than a baseline magnification on original UIs (mean 66.31 as reported by Mototaz et al. [55]).

We found a statistically significant difference (p = 0.0098, Z = 2.58) between the fullscreen and window mode; and between the window and fisheye mode (p = 0.026, Z = -2.22). However, there was no significant difference between the fullscreen and fisheye mode (p = 0.7, Z = 0.39). A Wilcoxon Signed-Rank test (two-tailed, $\alpha = 0.05$) was used in all three cases.

We discovered that frustration (mean 5.82, SD 2.14) was the most prominent factor in the high load indices of the fullscreen mode. This discovery is backed up by our general findings, as participants reported losing context and requiring to pan more in this mode. On the other hand, physical demand (mean 5.82, SD 2.82) was the main contributing factor to the fisheye mode's high score. This phenomenon is backed up by our study findings, as participants reported moving closer to the screen to understand heavily distorted content.

The overall score for the fisheye mode varied quite dramatically (min 18, max 98.67). Some participants (P4, P5) found the mode helpful, especially in T2. Consequently, they rated the mode to be less demanding. On the other hand, participants who sat closer to the display by default (§6.2.6), or had a medical condition that made distortions more distracting (P2, P3, P7, P8, P10), rated fisheye mode as very demanding.

6.2 Strategies and Preferences of Low-vision Users in Study Tasks

6.2.1 Navigation Strategy of Low-vision Users. We observed how our study participants searched the UIs for the overview task, T1, and the target-locating task, T2. For T1, almost all the participants (except P2 and P8) deployed a linear-like search strategy. In other words, they went through the contents in a left-to-right, top-to-bottom fashion. We noticed that they spent more time on text content when compared to graphical ones. When asked about this behavior, P3 said:

> "*I cannot seem to get the whole current word in my view (with my current level of magnification of around 500%). Thus, it takes more time to read texts.*"

P2 and P8, on the other hand, seemed to deploy a random peek-and-zoom strategy for T1. They tried to zoom into contrast-rich graphical items (e.g., buttons and images) at random and then guessed the purpose of the UI. However, we noticed that this often resulted in the participants missing out on critical objects of interest in the UI. For example, we included a baby's health progression application UI (consisting of metrics such as head size, weight, and height, among others) in T1. However, the word "`baby`" was not mentioned anywhere in the UI. There was only a tiny icon at the top showing a mother holding a baby. P2 and P8 missed out on reporting this information with their random peek-and-zoom search. Participants who deployed the former strategy (i.e., linear-like) were mostly able to report this information.

For task T2, the strategy for all participants was the same—zoom into likely targets based on the target description; if the zoomed-in object is not the target, move on to the next one. Unsurprisingly, the choice of the

next one was mainly based on its proximity to the current one. While some participants (P2, P4-P6, P8, P9-P11) relied more on the graphical description of the target (if available), others (P1, P3, P7) found the textual and graphical descriptions equally important. We also noticed that our participants took less time with targets that had both graphical and textual cues when compared to targets with only texts. Figure 12 demonstrates this finding. Figure 12 also displays the general improvement in the participants' performance when using the UIs from Space$_X$Mag's output. One minor exception was the (Graphics+Text) target in Window mode.

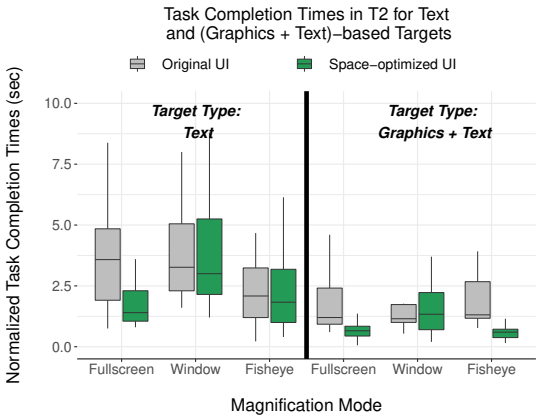


Fig. 12. Normalized task completion times in T2 based on the target types (e.g., Text and [Graphics + Text]). Lower is better. Different target types are demonstrated in Figure 7.

Some participants reported that the space-compacted UIs also impacted their search strategy. According to P1:

> "*I felt more confident navigating the space-optimized UIs as I was less afraid of losing information (context) on the screen. I also felt that the choice of magnification mode (among the three) was not as important with the space-optimized UIs*"

P3 also mentioned that he has to zoom less and can understand the context more easily with the space-optimized UIs.

6.2.2 Bounding Rectangles Around Objects is Helpful for Low-vision Users. Nine of our eleven participants said they found the bounding rectangles drawn around the UI objects helpful. According to P6:

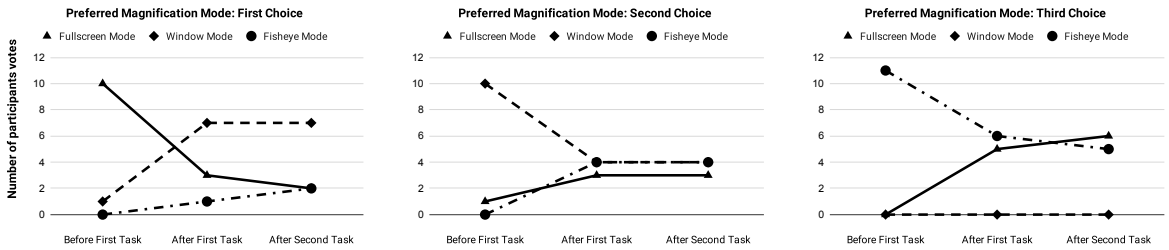
> "*Those (the red borders) are great! ZoomText does this, too (only for the item on focus), but it often messes up. These borders here are great in helping me to distinguish the objects and make sure I am not missing anything.*"

However, P5 and P10 believed that the borders could sometimes be distracting, especially when there are too many items on the screen.

6.2.3 Choice of Magnification Mode Depends on the Task at Hand. Almost all of our participants (except P9) were fullscreen magnification users. However, P9 used the window mode (in iOS Accessibility Zoom) predominantly. Moreover, only three of our participants were familiar with fisheye lenses. Unsurprisingly, when we asked them to rank the three magnification modes at the beginning of the study, ten of the eleven participants picked the fullscreen mode as their first choice (as shown in Figure 13a).

We took votes from the participants about their preferred magnification mode twice more—once after they finished the first task (not necessarily T1), and once when they were done with the second task. We found that

their rank of preference regarding magnification mode varied in each voting session, with window mode getting more popular and ending up as the preferred first choice for most at the end of the study (Figure 13a).



(a) First choice (b) Second choice (c) Third choice

Fig. 13. Magnification preferences of participants at different stages. Note that the first task is not necessarily T1.

This phenomenon demonstrates that the preferred magnification mode for LV users depends on their task at a particular time. We noticed that our participants (seven out of eleven) predominantly picked the window mode for reading texts, which is an integral part of the overview task, T1. Except for P4, no one else liked fisheye for reading texts—stating the distortion as the culprit.

For the target locating task (T2), however, participants were happy to pick either the window or fisheye, saying that it makes little difference. Nevertheless, they still voted for the window mode as their first choice. P1 and P7 stuck with their choice of fullscreen mode as their first choice throughout the study. Both mentioned convenience and being more familiar (with fullscreen mode) as the critical factors behind their decisions. We noticed that the fullscreen mode was most participants' third choice after the final voting session (Figure 13c)

6.2.4 Higher Magnification Does Not Always Guarantee Good Visibility. P2, P4, P8, and P10 pointed out that higher magnification is not always the only factor contributing to good visibility for a UI. P2 mentioned her struggles in UIs with light backgrounds, irrespective of the size of the UI elements. P4, P8, and P10 also mentioned their preference for a dark mode in a UI (i.e., light text on dark background). According to P10, the contrast ratio (higher the better), the choice of color for crucial objects (e.g., buttons, warnings), and the width (e.g., boldface or no-boldface) of the texts also play essential roles in improving the visibility of the contents in a UI for low-vision users.

6.2.5 Maintaining Original Aspect Ratio is Important for Text-rich UIs. Four participants (P2, P5, P6, P11) mentioned that they found the aspect ratio change in Space$_X$Mag's output discomforting. As the compacted UI was stretched to match the original shape (Figure 4b- 4c), the texts often became taller and more compact. This action made them harder to read for some participants. In the words of P5:

> *"Texts on the right (space-optimized UI) are harder to read because they are closer together. The alphabets look taller than they usually are."*

6.2.6 Fisheye's Distortion Bothers Those More Who Sit Closer to Their Displays. Participants who sat very close to the display (e.g., less than 2 inches or so), found the fisheye's distortion to be more bothering than those who sat at a more usual distance. For P2 and P7, this information regarding the sitting distance was self-reported. For P3, P8, and P10, we noticed the distance from their usage behavior during the study.

6.2.7 Preferred Lens Shape in Fisheye Mode. When using the Fisheye mode, our participants mostly preferred the rectangular shape. The summary of their choice is shown in Figure 14. They provided diversified rationales

for their choice. For example, P2 stated that the rectangle would allow more of a word to come into the current view for her to read. P3 felt that the rectangular-shaped lens created the least distortion among the three shapes (which is true as shown in Figures 5c and 5d). P6, P8, and P9 mentioned that the rectangular shape made reading much easier than the other two shapes.

While P6 felt that the oval shape might not be necessary, P3, P4, and P7 felt that the oval shape was a good middle ground with the circle and the rectangle on the two poles. Most of our participants reported problems with the circular shape as they felt it created the most distortion and brought more than one line into the view. However, P11 reported liking the circular shape the most. He felt that the higher distortion (in a circular shape) made the contents in the lens stand out more—something which P5 also agreed upon.

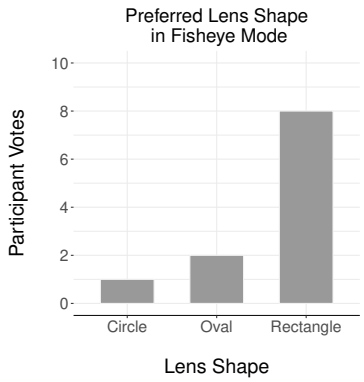


Fig. 14. Preferred lens shape in Fisheye mode as voted by the participants.

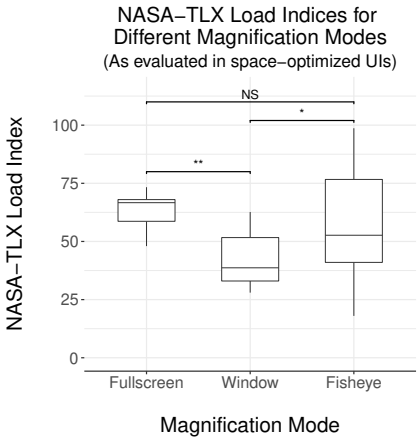


Fig. 15. NASA-TLX load indices as evaluated on Space$_X$Mag's outputs. Lower is better.

6.3 Alternative Feature Requests From Study Participants

6.3.1 Automatic Adaptation of the Magnification Mode, Magnification Amount, and Lens Shape. P6 and P11 suggested that it could be beneficial for low-vision users if their preferences (e.g., magnification mode, magnification amount, lens shape) can be applied to the UI at startup. P11 also suggested automatic adjustment of the lens shape and magnification type based on the current location of the lens (or focus). In his words:

> "*... another thing you can do is automatically adapt (of the magnification mode and lens shape). If the current item in focus is a graphical item, the fisheye lens with a circular/oval shape would be great. If we are on texts, switch back to window mode.*"

P3 believed automatic magnification control and lens size increment in some scenarios could be helpful. For example, if a low-vision user uses window mode for reading and the current word in focus is too big to fit in the lens, the system can automatically increase the lens size or reduce the magnification level.

6.3.2 Customizable Object Border Width and Color. P1, P5, P10, and P11 wished for customizable colors and widths for the bounding rectangles around the UI objects. P1 also believed that the use of these borders could be made optional.

6.3.3 Faster Panning. As we ran the study remotely using Zoom [10], there was often a minimal delay between the participants' actions (e.g., moving the lens, applying magnification) and the result of that action taking place on the screen. As such, they wished for reduced (or zero) delays when panning. Note that this delay was not noticeable on the local computer; if we could have run an in-person study, it is improbable that the participants would have noticed any delay.

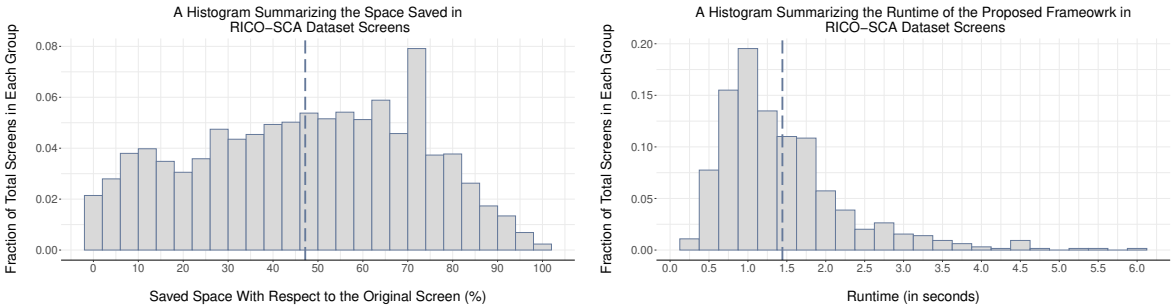
7 SCALABILITY OF SPACE$_X$MAG

To test the scalability and generalizability of Space$_X$Mag, we ran the framework on the RICO-SCA dataset [7] consisting of 25, 677 android screens. We successfully ran Space$_X$Mag on 16, 566 of those and generated space-compacted screens. The unsuccessful cases can be attributed to two possible reasons:

First, the accompanying `json` files that come with each screen contained ill-formed metadata information (e.g., overlapping objects, incorrect hierarchy).

Second, the optimizer (§3.3.1) failed to find a solution maintaining all the constraints (a limitation of GEKKO [20]).

7.1 Space Compaction Performance



(a) (b)

Fig. 16. Summary of Space$_X$Mag's performance on the RICO-SCA [7] dataset. (a) shows the % of space saved in each screen with respect to the original screen space. (b) shows the runtime of Space$_X$Mag for each screen. In both (a) and (b), the dashed line in the middle shows the mean.

The space-Compaction summary is shown in Figure 16a. We calculate the percentage of saved space, S_{saved} for a screen using the equation: $S_{\text{saved}} = \frac{\texttt{Area}(N_{org}) - \texttt{Area}(N_{opt})}{\texttt{Area}(N_{org})}$, where N_{org} and N_{opt} denote the root nodes in the UI hierarchy DAG in the original UI and the space-optimized UI. On average, we saved around 47.17% space (SD 24.58) for each UI screen in the database. We also noticed that the average space saving due to height reduction (mean 44.93, SD 23.83) was significantly higher than the space-saving due to width reduction (mean 5.56, SD 12.99). This output is understandable given that the mobile UI screens in the RICO-SCA dataset were already quite tall, and their widths were considerably smaller than their heights (aspect ratio 9 : 16).

7.2 Runtime

The mean runtime (i.e., time taken by Space$_X$Mag to generate output) for each screen was around 1.44 seconds (median 1.24, SD 0.83, min 0.21, max 6.06). The machine running the analysis had an 8-core CPU (Intel Core i7) and 32 Gigabytes of RAM. The runtime summary is shown in Figure 16b.

8 DISCUSSION

Our findings show positive outcomes for low-vision users when using Space$_X$Mag-generated user interfaces compared to conventional user interfaces. We now discuss the broader impact of Space$_X$Mag and provide design guidelines for system designers and accessibility practitioners.

8.1 Guidelines for System Designers and Accessibility Practitioners

8.1.1 Operating System-level Integration of Space$_X$Mag. To generate space-compacted UIs, Space$_X$Mag requires the meta-representation of the UI elements, and this representation is available in any operating system's graphics rendering pipeline. As such, Space$_X$Mag can be implemented in different operating systems (OS) such as Windows, Android, and iOS. Take the Android OS, for example. During the rendering phase of the graphics pipeline in Android [39], Image Stream Producers (e.g., media Player, camera preview) forward their specific window metadata to the *Window Manager*. *Window Manager* determines the windows' positions and forwards them to the SurfaceFlinger—which determines the order (e.g., foreground, background, occlusion) in which contents would be displayed on the screen. As such, we know that the *Window Manager* has the UI screen metadata regarding all the objects about to appear on the screen. Space$_X$Mag can be added as a routine in the *Window Manager*, who will forward the space-optimized window(s) to the SurfaceFlinger. Speaking from a high level, other operating systems (e.g., windows, iOS) also use a similar graphics pipeline (with possible implementation and security differences). As such, OS-level integration of Space$_X$Mag is feasible for any operating system.

8.1.2 Working With Space$_X$Mag When UI Meta-representation is Not Available. Recent Computer Vision-based methods have shown great potential in screen content detection, labeling, and creating hierarchies. Examples include Screen Recognition [83], Reflow [80], and Screen Parser [81] for iOS screens; UIED [82] and Screen2Words [79] for Android screens; and REMAUI [57] for both platforms. Additionally, there is the option of constructing UI metadata through reverse engineering of the UI pixels, as demonstrated in works such as Prefab [30, 31] and ScreenCrayons [60]. With these techniques, it is possible to generate the necessary UI metadata from a screenshot or image of a screen, even when the original metadata is unavailable.

8.1.3 Translating Keyboard and Mouse-based Interactions into Touch-based Interactions. The touch-based interactions used by current smartphone screen magnifiers have some major usability issues, despite being convenient. The panning and zooming gestures on smartphones are designed by overloading the standard touch gestures, such as one-finger touch or tap or drag. To pan, for instance, low-vision users need to drag the magnified screen with 3 fingers on iOS [3] or with 2 fingers on Android [2].

Recent work has identified a number of usability issues with these overloaded gestures. First, these gestures are cumbersome to use [24]; and quickly become tiring due to repeated use [74]. Second, these gestures require bimanual interaction, i.e., holding the smartphone with one hand while making the gestures with the other hand. Scenarios when another hand is encumbered [56], screen magnifiers could be challenging to use. Third, since all smartphone touch gestures involve some subset of finger combinations, it is easy to mix up one for the other. Fourth, panning with 3-finger-dragging is error-prone since not all three fingers touched the screen simultaneously [55]. The finger that touches the screen right before others registers an "unintended" one-finger tap event, which accidentally opens a different window. The recovery time from such accidental context switches varied widely, ranging from 5s to 90s [55].

As a result, any new magnification prototype must control several factors during evaluation. For instance, Momotaz et al. [55] evaluated their prototype by designing a specialized app that controlled (disabled) the interactivity of onscreen content. Similarly, the test UIs in our study setup were not interactive. Given our focus was on evaluating the visibility of UI content—as opposed to their interactivity, we believe this was a reasonable compromise. As for the mode of interaction, we offered primitive magnification operations through

mouse- and keyboard-based interactions, which allowed us to facilitate remote study. Contrary to most current screen magnifiers which offer uniform magnification only, our magnification prototype facilitates three modes of operation (e.g., fullscreen, window, fisheye), serving the varying needs of users. For the aforementioned reasons, we believe our prototype was reasonable for a controlled lab study; our findings can translate to real smartphone apps and will likely benefit low-vision users.

8.2 Placement of Space$_X$Mag Within Relevant Literature

8.2.1 Space$_X$Mag as a UI Adaptation Tool. We argue that Space$_X$Mag stands out in the larger body of UI adaptation tools because of its scalability, performance, platform neutrality, and user independence. To the best of our knowledge, none of the prior works offer all the aforementioned attributes at the same time. For instance, a noteworthy prior work, SUPPLE++ [38], put distinct attention to users with vision and motor disabilities when adapting UIs. It required user-specific information (e.g., visual cue size), was demonstrated on demo apps developed from scratch with no guidance on how to adapt an existing app, and took a significant amount of time to optimize a UI (e.g., $\approx$ 20 minutes in the worst scenario). Space$_X$Mag, on the other hand, requires no user-specific information, can be customized to run on any app with a graphical user interface (Section 8.1.1), and takes substantially less time ($\approx$ 6 seconds) even in the worst scenario.

8.2.2 Space$_X$Mag as a Platform-agnostic Differential Magnification Technique. Exisiting differential magnification techniques such as 'SteeringWheel' [22], 'Opportunistic Magnification' [21], and 'TableView' [51] are all proposed for web UIs, where an underlying content representation is readily available as an HTML DOM. Although the concept of an HTML DOM and a non-web application DOM are similar, their availability and quality differ drastically. For instance, the original RICO dataset [29] has around 72k screenshots from 9.7k Android applications, along with their DOMs. However, Li et al. [52] found major inconsistencies in about 75% of these DOMs, and created the filtered RICO-SCA [7] with around 25k screenshots. Unfortunately, we still found noteworthy inconsistencies (e.g., overlapping objects, incorrect hierarchy, repetition of containers) in the RICO-SCA dataset (Section 7). As such, fixing accessibility issues (e.g., redundant whitespace) for non-web apps pose a different challenge compared to web apps. This challenge warrants the need for platform-agnostic differential magnification tools. We argue that Space$_X$Mag addresses this need.

8.2.3 Space$_X$Mag as a Supplement for Existing Magnifiers. Popular magnifiers such as ZoomText [14], Windows Magnifier [9], and iOS Accessibility Zoom [3] mostly use uniform, fullscreen magnification. By reducing redundant, unnecessary whitespace prior to zooming, Space$_X$Mag makes life easy for these magnifiers, as the users are less likely to zoom into whitespace and lose context. This is also evident from our study findings as the participants reported not having to zoom as much with their magnification tools when using Space$_X$Mag-generated UIs (Section 6.1.1). As such, we argue that Space$_X$Mag enhances the capabilities of existing magnifiers.

8.3 Limitations and Future Work

Space$_X$Mag has several limitations. First, the framework can change the aspect ratio of the original UI, resulting from the unequal space savings in horizontal and vertical dimensions (§7). As observed in our study findings, this can make reading text contents difficult for some low-vision users. In the future, we plan to implement a non-uniform magnification on the resizing step (see Figure 4b- 4c) of Space$_X$Mag's optimization process. In this approach, we would aim to maintain the aspect ratios for at least some specific types of items (e.g., text and small icons).

Second, Space$_X$Mag-generated UIs are not always perfect (Appendix B.3). We noticed that Space$_X$Mag struggles specifically when there are modal and pop-up windows in the original screen (Figure 20(i)-(ii)). This is a limitation of the framework; by definition, we did not allow two UI objects to have overlapping coordinates unless they

had a child-parent relationship. However, modals are temporary UI segments and cannot be integrated into the original DOM tree. In the future, we plan to extend Space$_X$Mag to support modals and popups. Another scenario where Space$_X$Mag struggles is input UIs having an image or pattern as background (Figure 20(iii)-(iv)). As Space$_X$Mag only has access to the static image representation of the given UI, it cannot separate the foreground from the background (Section 3.3.4). However, in system-level integration, Space$_X$Mag would have access to such information thanks to *Window Manager* in the graphics rendering pipeline. Therefore, this is more of an issue with the input than the framework.

Third, we did not employ a formal strategy to evaluate the quality of all the Space$_X$Mag-generated UIs. While the preliminary feedback on the space-optimized UIs was predominantly positive (Section 6.1.1), the UIs used in the study represents a very small subset of the 16, 566 total UIs. As such, we plan to conduct a perceived quality study [72] on more of Space$_X$Mag's outputs in the future.

9 CONCLUSION

In this paper, we proposed Space$_X$Mag, an automatic framework to optimally compact whitespace in the UIs of mobile applications; to address the cumbersome experience (e.g., having to pan relentlessly, contents getting out of the viewport) low-vision users face when using different magnification tools at high magnification. We tested Space$_X$Mag in a study with 11 low-vision participants using three different magnification modes. We found that when using Space$_X$Mag-optimized UIs compared to original UIs, 11 low-vision participants in our study performed significantly better on two tasks. We tested the framework's scalability by running it on the RICO-SCA [7] dataset, and, on average, managed to save around 47.17% space in each Android screen.

Space$_X$Mag offers a set of attributes (i.e., scalability, performance, platform neutrality, and user independence) that no existing tool offers to the best of our knowledge. As such, we believe widespread operating system-level integration of Space$_X$Mag would enhance low-vision users' experience irrespective of their preferred magnification tools. We also argue that Space$_X$Mag will pave the way for more research in a similar direction, such as optimizing a UI hierarchy to save more space, fixing inconsistent alignments, and changing the relative size of different levels of UI content.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their insightful feedback. This work was supported in part by NIH subaward 87527/2/1159967. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

REFERENCES

[1] 2020. Android Magnification. https://support.google.com/accessibility/android/answer/6006949?hl=en
[2] 2020. Magnification – Android Accessibility Help. Retrieved October 8, 2020 from https://support.google.com/accessibility/android/answer/6006949?hl=en
[3] 2020. Zoom in on the iPhone screen. Retrieved October 8, 2020 from https://support.apple.com/guide/iphone/zoom-iph3e2e367e/ios
[4] 2022. Affine Transformation. https://en.wikipedia.org/wiki/Affine_transformation
[5] 2022. "Directed acyclic graph." Wikipedia, Wikimedia Foundation. en.wikipedia.org/wiki/Directed_acyclic_graph
[6] 2022. GEKKO optimization Suite. https://gekko.readthedocs.io/en/latest/
[7] 2022. RicoSCA. https://paperswithcode.com/dataset/ricosca
[8] 2022. Scale Matrix. https://en.wikipedia.org/wiki/Scaling_(geometry)
[9] 2022. Windows Magnifier. https://support.microsoft.com/en-us/topic/setting-up-and-using-magnifier-e1330ccd-8d5c-2b3c-d383-fd202808c71a
[10] 2022. Zoom. https://zoom.us/
[11] AFB. 2020. Glossary of Eye Conditions. http://www.afb.org/info/living-with-vision-loss/eye-conditions/12#L
[12] B. Agarwal and W. Stuerzlinger. 2013. WidgetLens: a system for adaptive content magnification of widgets. In *Proceedings of the 27th International BCS Human Computer Interaction Conference*. British Computer Society, 2578052, 1–10.

[13] B Agarwal and Wolfgang Stuerzlinger. 2013. WidgetLens: A system for adaptive content magnification of widgets. In *27th International BCS Human Computer Interaction Conference (HCI 2013) 27*. 1–10.

[14] Ai Squared. [n.d.]. ZoomText Magnifier. http://www.aisquared.com/zoomtext/more/zoomtext_magnifier.

[15] Ali S Alotaibi, Paul T Chiou, and William GJ Halfond. 2021. Automated repair of size-based inaccessibility issues in mobile applications. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 730–742.

[16] Android. [n.d.]. Accessibility Manager in Android. https://developer.android.com/reference/android/view/accessibility/AccessibilityManager

[17] AOA. [n.d.]. Common Types of Low Vision. https://www.aoa.org/patients-and-public/caring-for-your-vision/low-vision/common-types-of-low-vision

[18] Caroline Appert, Olivier Chapuis, and Emmanuel Pietriga. 2010. High-precision magnification lenses. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 273–282.

[19] Apple. [n.d.]. Debug Accessibility in iOS Simulator with the Accessibility Inspector. https://developer.apple.com/library/archive/technotes/TestingAccessibilityOfiOSApps/TestAccessibilityiniOSSimulatorwithAccessibilityInspector/TestAccessibilityiniOSSimulatorwithAccessibilityInspector.html.

[20] Logan DR Beal, Daniel C Hill, R Abraham Martin, and John D Hedengren. 2018. Gekko optimization suite. *Processes* 6, 8 (2018), 106.

[21] Jeffrey P. Bigham. 2014. Making the web easier to see with opportunistic accessibility improvement. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, 2647357, 117–122. https://doi.org/10.1145/2642918.2647357

[22] Syed Masum Billah, Vikas Ashok, Donald E. Porter, and I.V. Ramakrishnan. 2018. SteeringWheel: A Locality-Preserving Magnification Interface for Low Vision Web Browsing. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 3173594, 1–13. https://doi.org/10.1145/3173574.3173594

[23] John Brooke. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 194.

[24] Maria Claudia Buzzi, Marina Buzzi, Barbara Leporini, and Amaury Trujillo. 2015. Exploring Visually Impaired People's Gesture Preferences for Smartphones. In *Proceedings of the 11th Biannual Conference on Italian SIGCHI Chapter*. ACM, 2808448, 94–101. https://doi.org/10.1145/2808435.2808448

[25] Luca Cardelli. 1988. Building user interfaces by direct manipulation. In *Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software*. 152–166.

[26] Gutwin Carl and Skopik Amy. 2003. Fisheye Views are Good for Large Steering Tasks. In *CHI 2003 Conference on Human Factors in Computing Systems*. 201–208.

[27] M Sheelagh T Carpendale, David J Cowperthwaite, and F David Fracchia. 1995. 3-dimensional pliable surfaces: For the effective presentation of visual information. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*. 217–226.

[28] M Sheelagh T Carpendale, David J Cowperthwaite, and F David Fracchia. 1997. Making distortions comprehensible. In *Proceedings. 1997 IEEE Symposium on Visual Languages (Cat. No. 97TB100180)*. IEEE, 36–45.

[29] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 3126651, 845–854. https://doi.org/10.1145/3126594.3126651

[30] Morgan Dixon and James Fogarty. 2010. Prefab: Implementing Advanced Behaviors Using Pixel-based Reverse Engineering of Interface Structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. ACM, New York, NY, USA, 1525–1534. https://doi.org/10.1145/1753326.1753554

[31] Morgan Dixon, Alexander Nied, and James Fogarty. 2014. Prefab layers and prefab annotations: extensible pixel-based interpretation of graphical interfaces. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. 221–230.

[32] Peitong Duan, Casimir Wierzynski, and Lama Nachman. 2020. Optimizing user interface layouts via gradient descent. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–12.

[33] James Fogarty and Scott E Hudson. 2003. GADGET: A toolkit for optimization-based approaches to interface and display generation. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*. 125–134.

[34] Julie Fraser and Carl Gutwin. 2000. A framework of assistive pointers for low vision users. In *Proceedings of the fourth international ACM conference on Assistive technologies*. ACM, 354329, 9–16. https://doi.org/10.1145/354324.354329

[35] Krzysztof Gajos and Daniel S. Weld. 2004. SUPPLE: Automatically Generating User Interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces* (Funchal, Madeira, Portugal) *(IUI '04)*. Association for Computing Machinery, New York, NY, USA, 93–100. https://doi.org/10.1145/964442.964461

[36] Krzysztof Z Gajos, Daniel S Weld, and Jacob O Wobbrock. 2008. Decision-Theoretic User Interface Generation.. In *AAAI*, Vol. 8. 1532–1536.

[37] Krzysztof Z Gajos, Daniel S Weld, and Jacob O Wobbrock. 2010. Automatically generating personalized user interfaces with Supple. *Artificial intelligence* 174, 12-13 (2010), 910–950.

[38] Krzysztof Z Gajos, Jacob O Wobbrock, and Daniel S Weld. 2007. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. 231–240.

[39] Google. 2022. Android Graphics. https://source.android.com/docs/core/graphics#android-graphics-components

[40] Google. 2022. Building Layouts with an Adapter. https://developer.android.com/develop/ui/views/layout/declaring-layout#AdapterViews

[41] Carl Gutwin. 2002. Improving focus targeting in interactive fisheye views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 503424, 267–274. https://doi.org/10.1145/503376.503424

[42] Carl Gutwin. 2002. Improving focus targeting in interactive fisheye views. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 267–274.

[43] Carl Gutwin and Amy Skopik. 2003. Fisheyes are good for large steering tasks. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 201–208.

[44] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.

[45] Kasper Hornbæk, Benjamin B Bederson, and Catherine Plaisant. 2002. Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM Transactions on Computer-Human Interaction (TOCHI)* 9, 4 (2002), 362–389.

[46] Kasper Hornbæk and Erik Frøkjær. 2001. Reading of electronic documents: the usability of linear, fisheye, and overview+detail interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 365118, 293–300. https://doi.org/10.1145/365024.365118

[47] Won Chul Kim and James D Foley. 1993. Providing high-level control and expert assistance in the user interface presentation design. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*. 430–437.

[48] Richard L. Kline and Ephraim P. Glinert. 1995. Improving GUI accessibility for people with low vision. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press/Addison-Wesley Publishing Co., 223919, 114–121. https://doi.org/10.1145/223904.223919

[49] John Lamping, Ramana Rao, and Peter Pirolli. 1995. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press/Addison-Wesley Publishing Co., 223956, 401–408. https://doi.org/10.1145/223904.223956

[50] Hae-Na Lee and Vikas Ashok. 2022. Customizable Tabular Access to Web Data Records for Convenient Low-vision Screen Magnifier Interaction. *ACM Transactions on Accessible Computing (TACCESS)* 15, 2 (2022), 1–22.

[51] Hae-Na Lee, Sami Uddin, and Vikas Ashok. 2020. TableView: Enabling efficient access to web data records for screen-magnifier users. In *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility*. 1–12.

[52] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile UI action sequences. *arXiv preprint arXiv:2005.03776* (2020).

[53] James Lin and James A Landay. 2002. Damask: A tool for early-stage design and prototyping of multi-device user interfaces. In *In Proceedings of The 8th International Conference on Distributed Multimedia Systems (2002 International Workshop on Visual Computing)*. Citeseer, 573–580.

[54] Nesrine Mezhoudi. 2013. User interface adaptation based on user feedback and machine learning. In *Proceedings of the companion publication of the 2013 international conference on Intelligent user interfaces companion*. 25–28.

[55] Farhani Momotaz and Syed Masum Billah. 2021. Tilt-Explore: Making Tilt Gestures Usable for Low-Vision Smartphone Users. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY, USA, 1154–1168. https://doi.org/10.1145/3472749.3474813

[56] Alexander Ng, Stephen A. Brewster, and John H. Williamson. 2014. Investigating the Effects of Encumbrance on One- and Two- Handed Interactions with Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. ACM, New York, NY, USA, 1981–1990. https://doi.org/10.1145/2556288.2557312

[57] Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse engineering mobile application user interfaces with remaui (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 248–259.

[58] Jeffrey Nichols, Brad A Myers, Michael Higgins, Joseph Hughes, Thomas K Harris, Roni Rosenfeld, and Mathilde Pignol. 2002. Generating remote control interfaces for complex appliances. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*. 161–170.

[59] Chris North and Ben Shneiderman. 2000. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *Proceedings of the working conference on Advanced visual interfaces*. 128–135.

[60] Dan R Olsen Jr, Trent Taufer, and Jerry Alan Fails. 2004. ScreenCrayons: annotating anything. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*. 165–174.

[61] Emmanuel Pietriga and Caroline Appert. 2008. Sigma lenses: focus-context transitions combining space, time and translucence. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1343–1352.

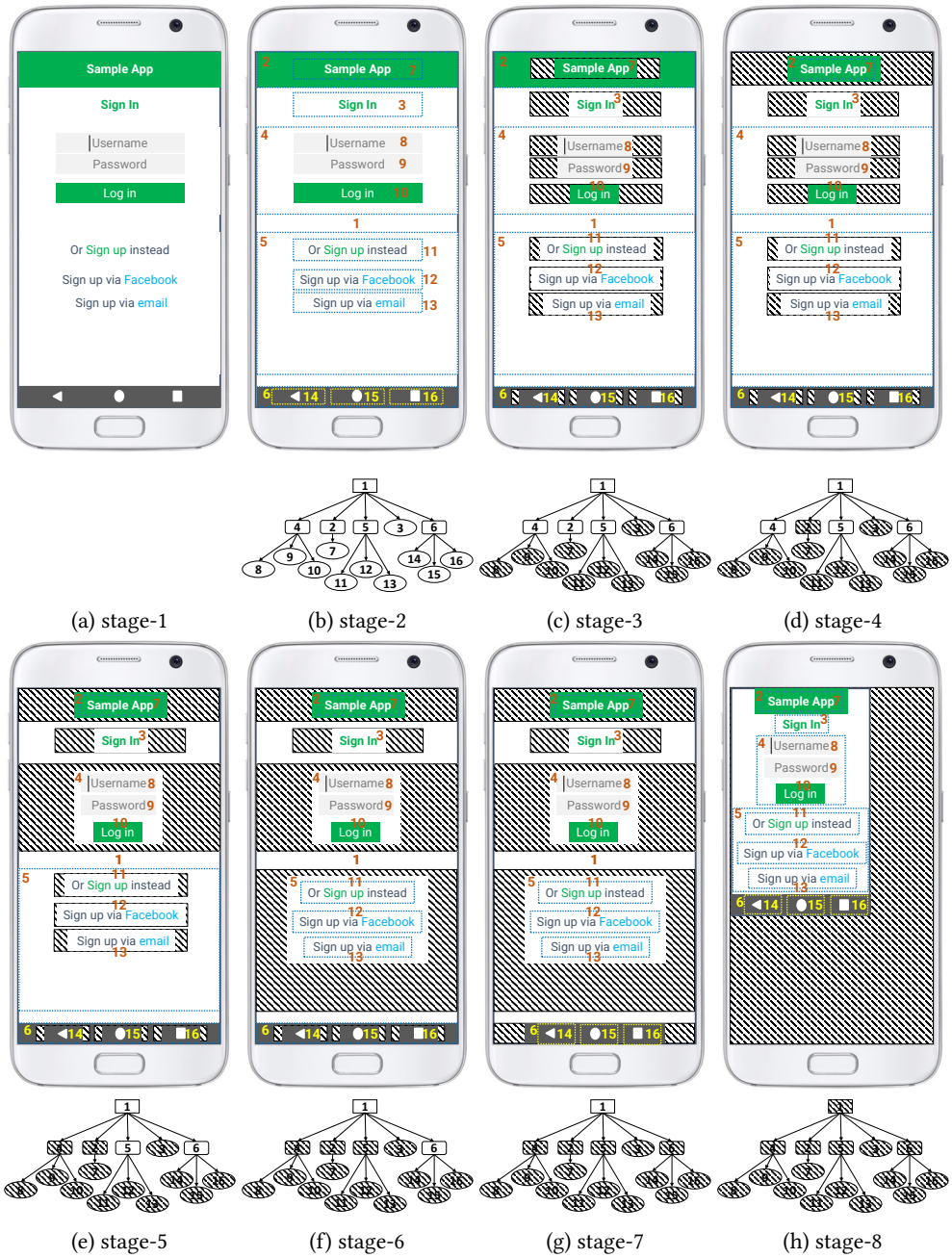
[62] Emmanuel Pietriga, Caroline Appert, and Michel Beaudouin-Lafon. 2007. Pointing and beyond: an operationalization and preliminary evaluation of multi-scale searching. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1215–1224.

[63] Catherine Plaisant, David Carr, and Ben Shneiderman. 1995. Image-browser taxonomy and guidelines for designers. *Ieee Software* 12, 2 (1995), 21–32.

[64] Shankar R Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. 2001. ICrafter: A service framework for ubiquitous computing environments. In *International Conference on Ubiquitous Computing*. Springer, 56–75.
[65] Angel Puerta and Jacob Eisenstein. 2002. XIML: A universal language for user interfaces. *White paper* (2002).
[66] Mahfuz Rahman, Sean Gustafson, Pourang Irani, and Sriram Subramanian. 2009. Tilt Techniques: Investigating the Dexterity of Wrist-based Input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. ACM, New York, NY, USA, 1943–1952. https://doi.org/10.1145/1518701.1518997
[67] Gonzalo Ramos, Andy Cockburn, Ravin Balakrishnan, and Michel Beaudouin-Lafon. 2007. Pointing lenses: facilitating stylus input through visual-and motor-space magnification. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 757–766.
[68] George G Robertson and Jock D Mackinlay. 1993. The document lens. In *Proceedings of the 6th annual ACM symposium on User interface software and technology*. 101–108.
[69] Manojit Sarkar and Marc H Brown. 1994. Graphical fisheye views. *Commun. ACM* 37, 12 (1994), 73–83.
[70] Manojit Sarkar, Scott S Snibbe, Oren J Tversky, and Steven P Reiss. 1993. Stretching the rubber sheet: A metaphor for viewing large layouts on small screens. In *Proceedings of the 6th annual ACM symposium on User interface software and technology*. 81–91.
[71] Jeff Sauro. 2018. 5 Ways to Interpret a SUS Score. https://measuringu.com/interpret-sus-score/
[72] Morteza Shiripour, Niraj Ramesh Dayama, and Antti Oulasvirta. 2021. Grid-based Genetic Operators for Graphical Layout Generation. *Proceedings of the ACM on Human-Computer Interaction* 5, EICS (2021), 1–30.
[73] Pedro Szekely. 1996. Retrospective and challenges for model-based interface development. In *Design, Specification and Verification of Interactive Systems' 96*. Springer, 1–27.
[74] Sarit Felicia Anais Szpiro, Shafeka Hashash, Yuhang Zhao, and Shiri Azenkot. 2016. How People with Low Vision Access Computing Devices: Understanding Challenges and Opportunities. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, 2982168, 171–180. https://doi.org/10.1145/2982142.2982168
[75] John J Taylor, Rachel Bambrick, Andrew Brand, Nathan Bray, Michelle Dutton, Robert A Harper, Zoe Hoare, Barbara Ryan, Rhiannon T Edwards, Heather Waterman, et al. 2017. Effectiveness of portable electronic and optical magnifiers for near vision activities in low vision: a randomised crossover trial. *Ophthalmic and Physiological Optics* 37, 4 (2017), 370–384.
[76] Mary Frances Theofanos and Janice Redish. 2005. Helping Low-vision and Other Users with Web Sites That Meet Their Needs: Is One Site for All Feasible? *Technical Communication* 52, 1 (2005), 9–20. http://www.ingentaconnect.com/content/stc/tc/2005/00000052/00000001/art00002
[77] Daniel W Turner III and Nicole Hagstrom-Schmidt. 2022. Qualitative interview design. *Howdy or Hello? Technical and Professional Communication* (2022).
[78] Andreas Wächter and Lorenz T Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* 106, 1 (2006), 25–57.
[79] Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. 2021. Screen2words: Automatic mobile UI summarization with multimodal learning. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 498–510.
[80] Jason Wu, Titus Barik, Xiaoyi Zhang, Colin Lea, Jeffrey Nichols, and Jeffrey P Bigham. 2022. Reflow: Automatically Improving Touch Interactions in Mobile Applications through Pixel-based Refinements. *arXiv preprint arXiv:2207.07712* (2022).
[81] Jason Wu, Xiaoyi Zhang, Jeff Nichols, and Jeffrey P Bigham. 2021. *Screen Parsing: Towards Reverse Engineering of UI Models from Screenshots*. Association for Computing Machinery, New York, NY, USA, 470–483. https://doi.org/10.1145/3472749.3474763
[82] Mulong Xie, Sidong Feng, Zhenchang Xing, Jieshan Chen, and Chunyang Chen. 2020. UIED: a hybrid tool for GUI element detection. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1655–1659.
[83] Xiaoyi Zhang, Lilian de Greef, Amanda Swearngin, Samuel White, Kyle Murray, Lisa Yu, Qi Shan, Jeffrey Nichols, Jason Wu, Chris Fleizach, et al. 2021. Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

A STEP-BY-STEP DEMONSTRATION OF SPACE$_X$MAG'S SPACE OPTIMIZATION

In this section, we provide a step-by-step demonstration of how Space$_X$Mag's space optimization works. Figure 17 demonstrates how a bottom-up space optimization is performed in a sample application UI. Figure 17(a) shows the original application UI. Figure 17(b) numbers all the objects in the UI and creates a corresponding DAG below. All nodes' coordinates (i.e., optimization parameters) are known at this point. The parent-child relationships (i.e., a model constraint) are also known and shown using the directed edges in the DAG. For example, nodes 11, 12, 13 are the children of node 5. Other optimization constraints are also detected at this stage. A relative position constraint example is that node 12 is above 13 but below 11. All such constraints are respected throughout the optimization



(a) stage-1 (b) stage-2 (c) stage-3 (d) stage-4

(e) stage-5 (f) stage-6 (g) stage-7 (h) stage-8

Fig. 17. Different stages in Space$_X$Mag's optimization. Each stage shows a UI (top) and its relevant DAG (bottom). Saved whitespace for a node is shown using an **angled-line** pattern.

process, from Stage 3 to Stage 8. At this stage, none of the UI objects is space-optimized—thus, the color of their corresponding nodes in the associated DAG is white. Finally, the DAG's traversal order for the nodes is determined using a *post-order* traversal. The order shown in this example is: {7, 3, 8, 9, 10, 11, 12, 13, 14, 15, 16, 2, 4, 5, 6, 1}.

Stage 3 optimizes all the atomic objects in the UI. Note that most atomic objects in UIs reside in an intermediate container. While atomic objects can sometimes have the same bounding rectangle as their containers, most of the time, they do not. This phenomenon is evident in our analysis of the RICO-SCA [7] dataset. Here, we assume that the atomic objects reside in containers that have larger bounding rectangles. As such, we optimize the space inside these containers and eliminate them from our consideration. Note that the containers are not shown in the DAG. As an object gets optimized, the space reduced in the process is shown using an *angled-line* pattern. The DAG node corresponding to this UI object is marked in the same pattern.

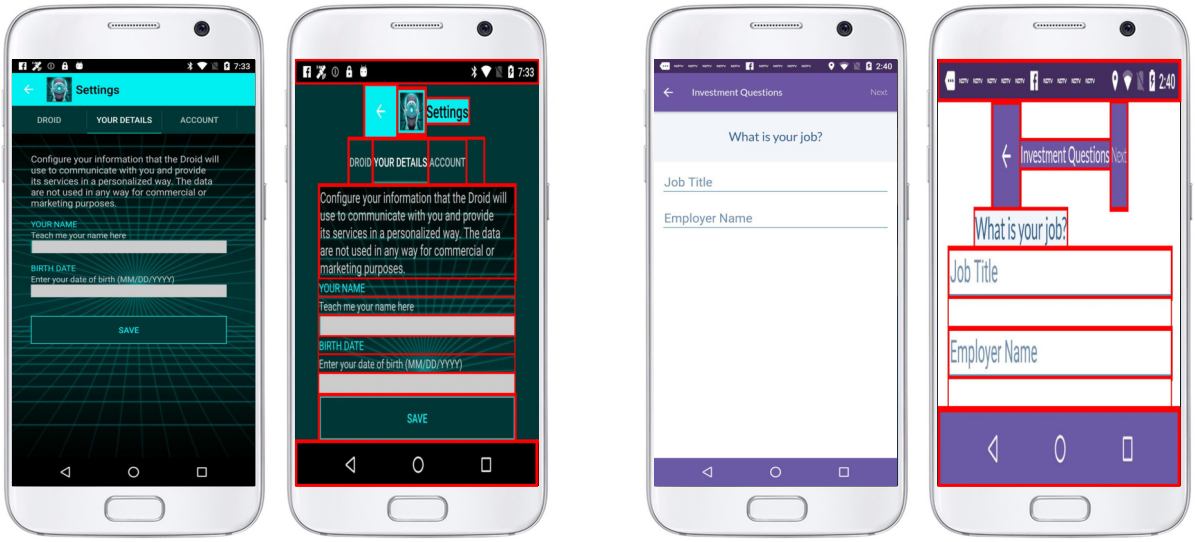
Stage 4 optimizes the first parent node in the DAG (i.e., node 2). Note that after this optimization, the previously saved space from node 7 and the current saved space from node 2 are merged. Stage 5-7 optimizes nodes 4, 5, and 6, respectively. Finally, stage 8 optimizes node 1 and generates an optimal meta-representation for the UI.

The original UI has a central alignment, which is respected throughout the optimization process. However, a specific alignment is often hard to detect for real-world applications. In such scenarios, Space$_X$Mag applies a central alignment.

B SPACE$_X$MAG-GENERATED SCREENSHOTS GALLERY

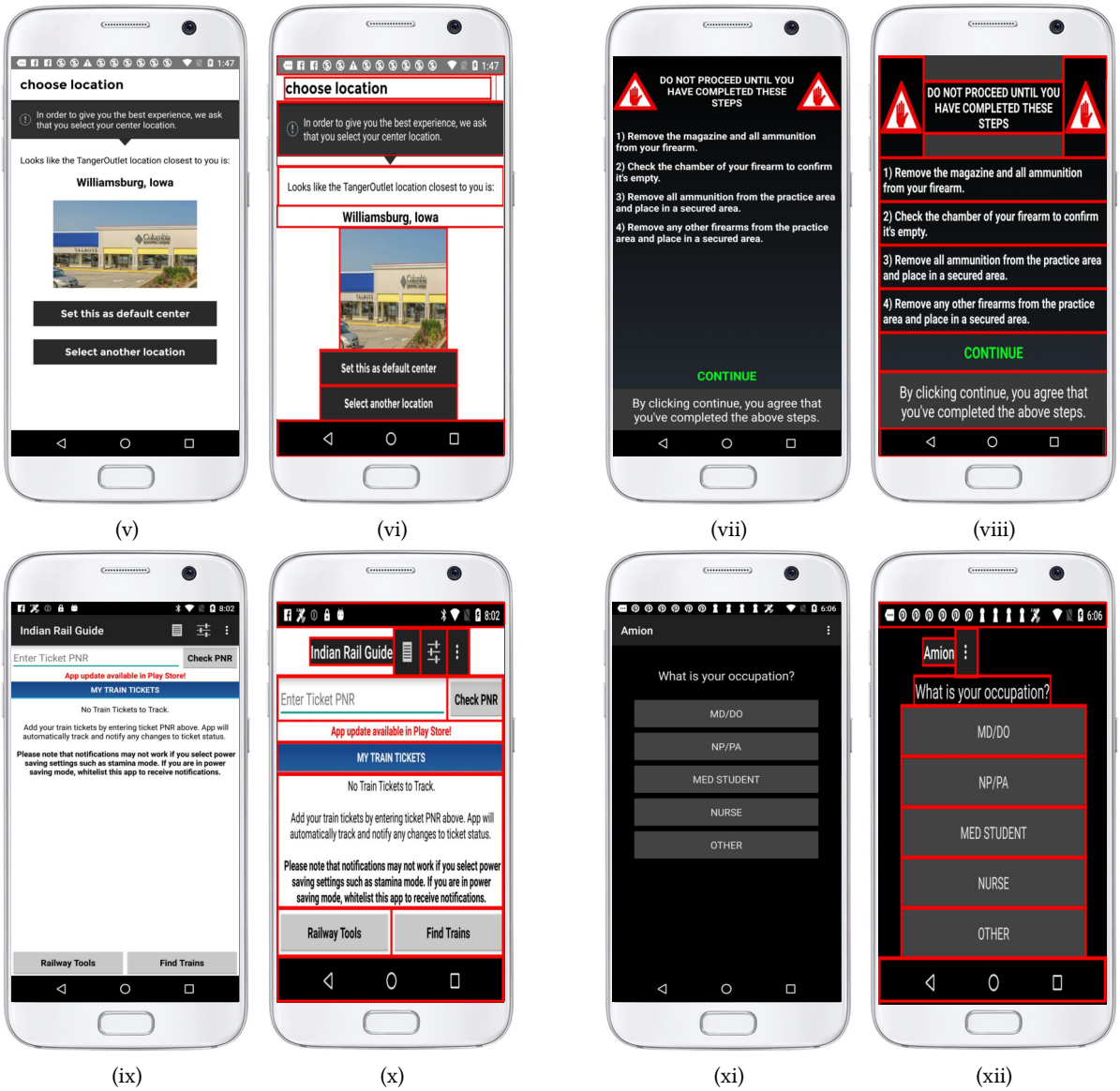
In this section, we present some examples of the Space$_X$Mag-generated UIs, organized in three groups: screenshots with ample whitespace, screenshots with little whitespace, and failed/partially failed samples.

B.1 Screenshots With Ample Whitespace



(i) (ii) (iii) (iv)

Fig. 18. Space$_X$Mag-generated UIs when input UIs have ample whitespace.



(v) (vi) (vii) (viii)

(ix) (x) (xi) (xii)

Fig. 18. Space$_X$Mag-generated UIs when input UIs have ample whitespace (contd.). Space$_X$Mag usually works the best when the input UI has ample space.

B.2 Screenshots With Little Whitespace

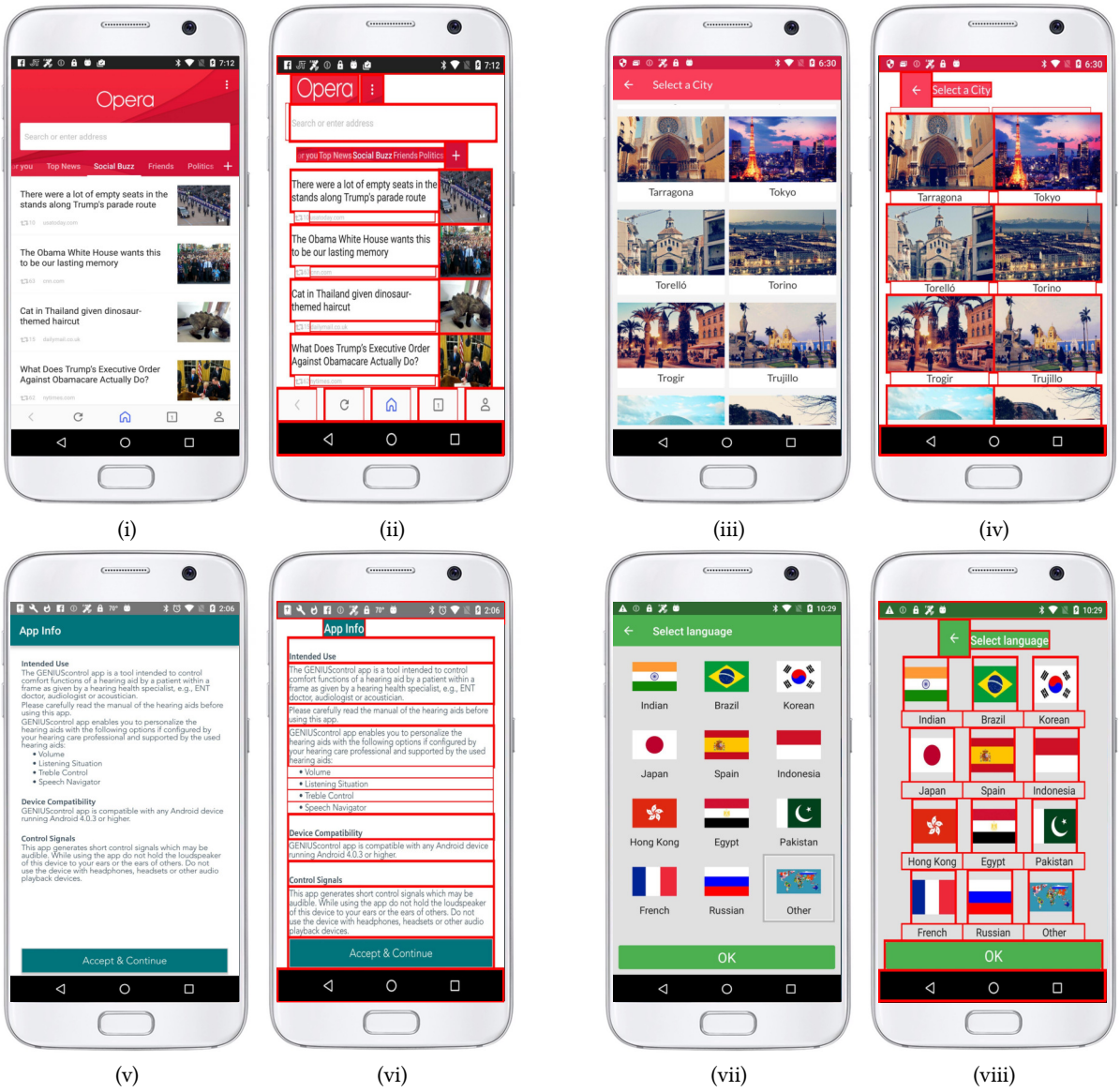


Fig. 19. In compact UIs, Space$_X$Mag has little to do. Importantly, the framework does not break because of the complexity of the UI.

B.3 Failed/Partially Failed Samples

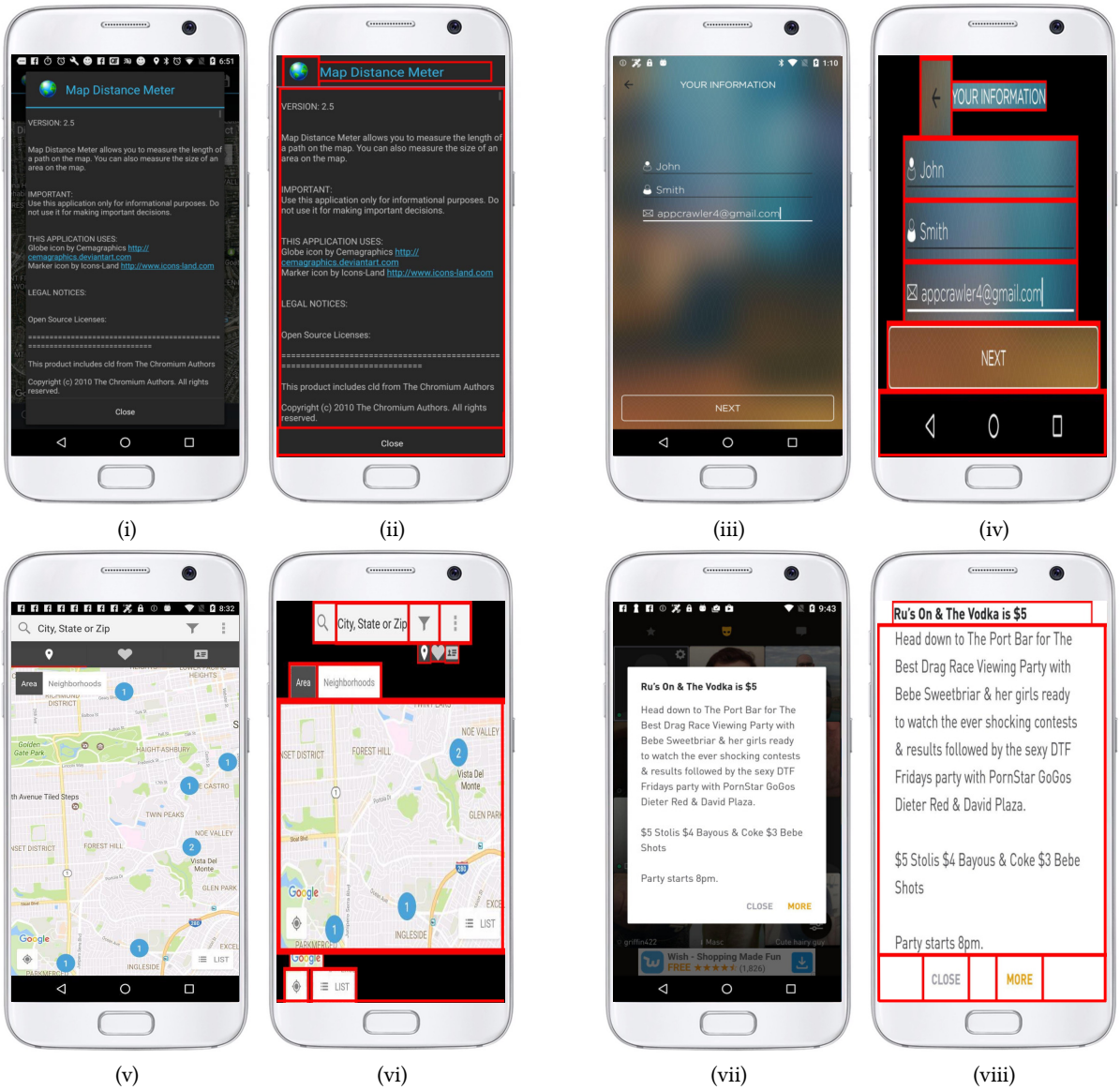
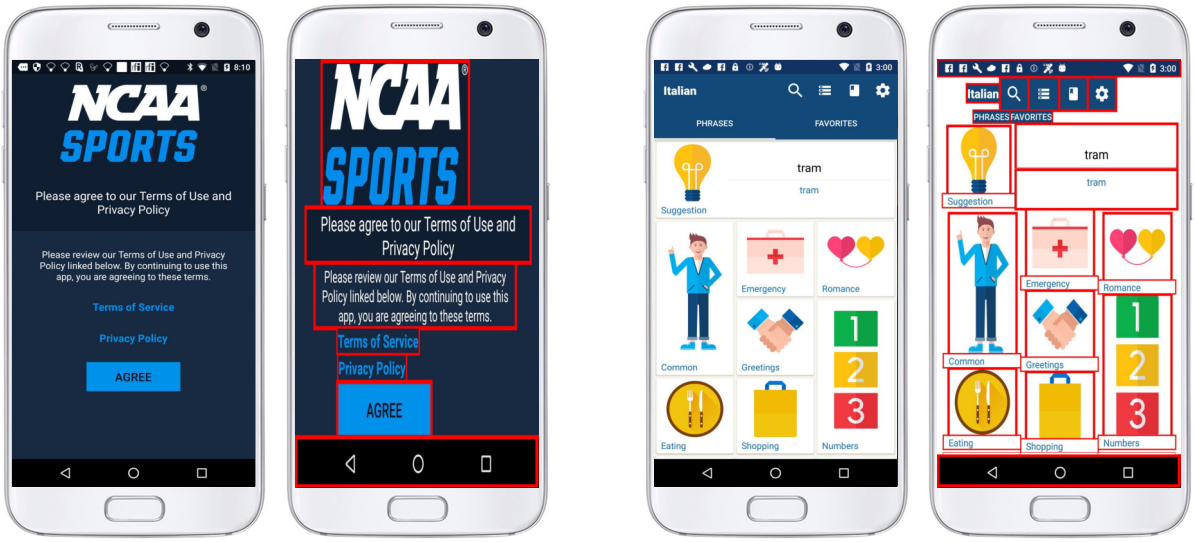


Fig. 20. Examples where Space$_X$Mag fails to generate a consistent UI as output.
(i)→(ii): Only the popup is detected and rendered, ignoring the primary UI in the background. *Probable Cause*: The DOM contained only the popup's information.
(iii)→(iv): The background color/pattern is lost in the final rendering. *Probable Cause*: Space$_X$Mag's current implementation cannot separate the foreground from the background (Section 8.3).
(v)→(vi): Misplacement/overlap of UI objects in the final rendering. *Probable Cause*: Incorrect DOM information.
(vii)→(viii): Same issue as (i)→(ii).



(ix) (x) (xi) (xii)

Fig. 20. Examples where Space$_X$Mag fails to generate a consistent UI as output (contd.).
(ix)→(x): Loss of original alignment. *Probable Cause*: The optimizer (i.e., GEKKO) got stuck in a local minimum.
(xi)→(xii): Too many red rectangles make the UI look more compact than the original. *Probable Cause*: Space$_X$Mag draws a red discernible rectangle around every UI object; sometimes it can be too much.