SalusChart: A Data Visualization Library for Mobile Health Apps

Sebin Hwang* Yonsei University
Seongjae Bae† Yonsei University
Jehu Ahn‡ Yonsei University
Zekai Shao§ Fudan University
Eun Kyoung Choe¶ University of Maryland, College Park
Bongshin Lee‖ Yonsei University

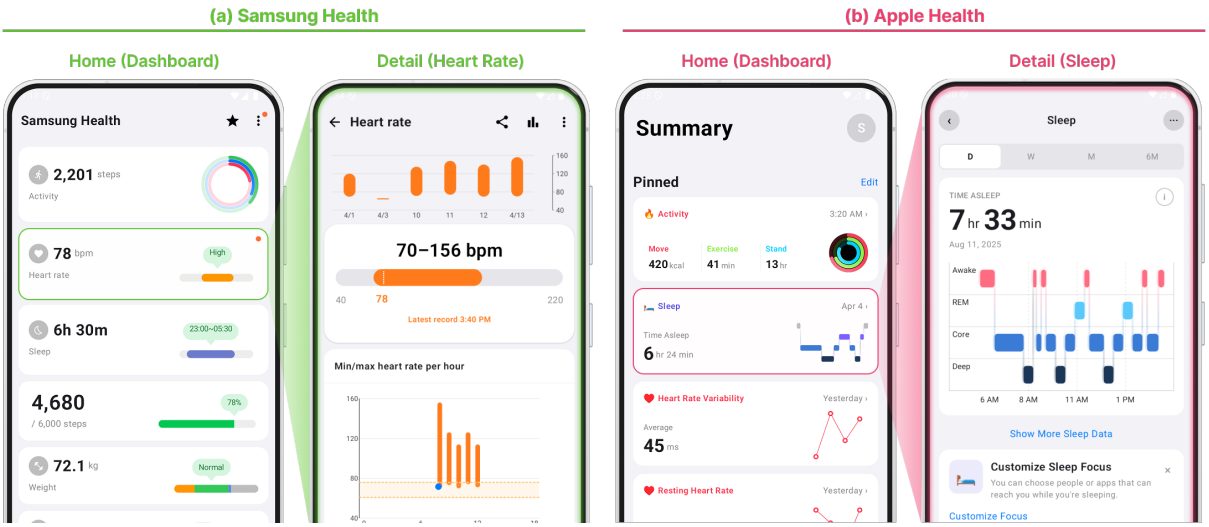

Figure 1: Example mHealth app screens reproduced using SalusChart: (a) Samsung Health-style dashboard and heart-rate detail views, and (b) Apple Health-style dashboard and sleep detail views.

Abstract

Data visualization is an essential component of mobile health (mHealth) apps for delivering users' personal health data. However, creating effective visualizations for mobile and wearable devices is challenging: screen space is limited, touch interaction is coarse, and developers may lack the visualization expertise needed to design for small screens and non-expert users. Existing general-purpose chart libraries compound this difficulty by ignoring health-specific contexts, leaving each developer to make independent design decisions about how health data should be presented. The result is inconsistent and often poorly designed visualizations across apps. We present SalusChart, an open-source Android visualization library built with Jetpack Compose, specifically designed for mHealth apps. Its reusable components encode visualization design principles and support the development of more consistent visualizations across mHealth apps on mobile and wearable devices.

Index Terms: Mobile data visualization, Visualization library, Mobile health, Personal health.

*e-mail: ssebin@yonsei.ac.kr
†e-mail: jae7151@yonsei.ac.kr
‡e-mail: ilop0624@gmail.com
§e-mail: gemini25szk@gmail.com
¶e-mail: choe@umd.edu
‖e-mail: b.lee@yonsei.ac.kr
Sebin Hwang and Seongjae Bae are co-first authors.
Eun Kyoung Choe and Bongshin Lee are co-corresponding authors.

1 Introduction

Mobile health (mHealth) apps commonly use visualizations to communicate progress, goals, and changes in users' personal health data [11, 13]. However, developing effective visualizations for mHealth apps presents significant challenges. Mobile interfaces are constrained by limited screen space and on-the-go use [12, 22], and also need to accommodate touch-friendly interaction [19]. Furthermore, the current mobile development ecosystem often lacks native support for the visualization design principles and domain-specific capabilities needed for health data contexts [18].

mHealth app developers currently rely on general-purpose libraries that lack health-specific needs, such as annotations for clinically meaningful ranges or data navigation across varying time scales. As a result, developers often implement common patterns from scratch, leading to inconsistent visualizations across mHealth apps. In addition, many mobile app developers may have limited exposure to the visualization principles to design representations suited to small-screen, everyday use by lay individuals. This can result in visual clutter, poor readability, and ambiguous encodings that hinder accurate interpretation of health data [18].

To address these challenges, we present SalusChart, a mobile data visualization library built for mHealth apps. To inform its design and implementation, we analyzed recurring visualization patterns in widely used mHealth apps, including Samsung Health and Apple Health. Guided by four core design goals that build on design considerations by Lee et al. [18], we implemented SalusChart as an Android library using Jetpack Compose and released as open source (`https://github.com/HDIL-YS/SalusChart`).

To demonstrate its applicability, we reproduced selected views in existing health apps with SalusChart. Beyond bundling common chart types, SalusChart encodes health-specific visualization patterns, defaults, and annotations into reusable components, reducing

repetitive implementation effort and supporting more consistent visualizations across mHealth apps.

2 Related Work

Mobile Visualization for Health. Small screens, finger-based touch input, and the absence of hover-based interaction affect both readability and interaction, making many common desktop visualization patterns unsuitable for direct transfer to mobile settings [20, 21]. Effective mobile visualization therefore requires both layout and interaction to be designed for mobile form factors and usage contexts [14].

Prior work on personal data exploration on smartphones also demonstrates that mobile interaction techniques, such as touch and speech, influence how users navigate and interpret personal data visualizations [17]. These mobile design challenges are especially acute in health contexts. Users may want quick, glanceable feedback in the moment, while also wanting longer-term reflection on past behaviors and outcomes. Meyer et al. describe this spectrum as ranging from "my health, now" to "my health, in the past" [22].

Recent studies of deployed mHealth apps show that health visualizations continue to face usability issues related to interaction, functionality, and consistency across metrics [10, 11, 13]. Beyond usability, mHealth visualizations also need to support interpretation by a broad user population. Kim argues that self-generated health data is continuously collected in everyday life and needs to be understandable to non-expert users with varying levels of health and visualization literacy [16], echoing broader discussions in visualization research on designing for diverse audiences beyond expert analysts [15, 20]. More recently, Lee et al. consolidated these concerns into a set of design considerations for dedicated mHealth visualization libraries, including intelligent defaults, health-specific annotations, fluid interaction, temporal structure, glanceable representations for smaller screens, and inclusive design [18]. Guided by these considerations, we design and develop a mobile data visualization library for health.

Mobile Visualization Libraries and Toolkits. Several chart libraries are already available for mobile app development. On Android, libraries such as MPAndroidChart [6], HelloCharts [5], and AnyChart [1] provide general-purpose chart views with interactive features including zooming, panning, and selection across common chart types. More recently, libraries such as Vico [8] and YCharts [9] have introduced Compose-based chart components, reflecting the growing adoption of declarative UI development on Android. On iOS, developers rely on toolkits such as Charts [3] and Swift Charts [7].

As general-purpose libraries, they primarily expose primitives such as bars, lines, axes, and points, leaving the developers to handle composition and interpretation. Without opinionated defaults, developers make independent decisions about axis scales, color encodings, and chart density that can lead to truncated axes, ambiguous encodings, and charts too dense for small screens [10]. Beyond defaults, recurring chart types in mHealth apps such as range charts and sleep stage charts are difficult to build from generic primitives without custom Canvas implementations. Existing libraries provide primitives for these features, but do not package them as domain-aware components. This leaves a gap for libraries that combine the flexibility of general-purpose toolkits with the constraints and conventions needed for well-designed mHealth visualizations.

3 SalusChart

3.1 Design Goals

We designed SalusChart around four goals (G1–G4), drawn from considerations for mHealth visualization libraries by Lee et al. [18]. We translated a subset of these high-level considerations into concrete, implementation-oriented goals that shaped SalusChart's architecture and API. Specifically, G1 is informed by *Treating Time as a First-Class Citizen*, G2 by *Support for Smaller and Variable Screens*, G3 by *Annotations for Contextual Information and Health Semantics*, and G4 by *Intelligent Default Representations*.

G1: Support for Health-Specific Data Structures. Health data are structurally different from generic time-series data. Sleep data have ordered stages with durations, vital signs are often represented as ranges rather than single values, and activity may involve multiple attributes. Approximating these with generic primitives forces developers to write custom Canvas code and loses underlying data semantics. Health data are also longitudinal, so visualizations must allow users to navigate time while keeping axes stable. SalusChart represents these as a small set of typed marks built from a single time-indexed dataset, rather than leaving each chart to handle raw health records with custom code.

G2: Handling of Two Levels of Detail. Health dashboards typically present many metrics at once as card-style summaries, where showing full detail can lead to clutter and reduced readability. These summaries therefore need minimal chart representations for quick overviews, while the same metrics benefit from full representations in detail views. We observed this pattern across the mHealth apps we surveyed, motivating support for paired minimal and full representations within a single framework.

G3: Native Annotation Support for Health Semantics. Health visualizations rely on annotations to convey clinically meaningful information, such as normal ranges, goals, and clinical thresholds. Providing these natively spares developers from ad hoc overlays and repetitive custom code. Beyond developer-specified thresholds, the library should also compute data-driven annotations such as averages and trend lines. In addition, annotations should remain anchored to corresponding data items during paging or scrolling, rather than to fixed pixel positions.

G4: Intelligent Defaults with Configurable Overrides. Health metrics follow visual conventions that users are already familiar with: red is commonly used to represent heart rate, while purple frequently denotes sleep in platforms such as Samsung Health and Apple Health. Intelligent defaults grounded both in these conventions and in visualization design principles can promote easy interpretation and reduce design effort, helping developers without visualization expertise produce reasonable charts out of the box. A library should provide built-in conventions, such as named styles and default encodings, while allowing overrides at both the app and chart levels, so developers can adapt visualizations to their own branding, accessibility requirements, or product needs.

3.2 Identifying Chart Types for mHealth Visualization Library

To determine SalusChart's initial scope, we surveyed chart types across four mainstream mobile health apps—Samsung Health, Apple Health, Google Fit, and Fitbit—in November 2025. The first three authors independently coded the chart types and resolved disagreements through discussion. We considered a chart type distinct when it used a unique visual mark or encoding rather than a combination of existing types; composite charts that overlay existing types on a shared axis, such as a bar chart with a line chart, were therefore counted under their constituent types. This yielded 17 distinct chart types, with Samsung Health and Apple Health offering the most extensive and visually sophisticated set and Google Fit and Fitbit offering fewer distinct types.

From these, we prioritized chart types that generalize across metrics and excluded four: ECG waveforms and menstruation logs, which are highly domain-specific; maps, which are spatial rather than temporal; and Samsung's heart-shaped ring, an app-specific, branded form that we differentiated from general activity rings. The remaining 13 chart types (Table 1) align with the need for health-specific primitives (G1) and compact summaries (G2).

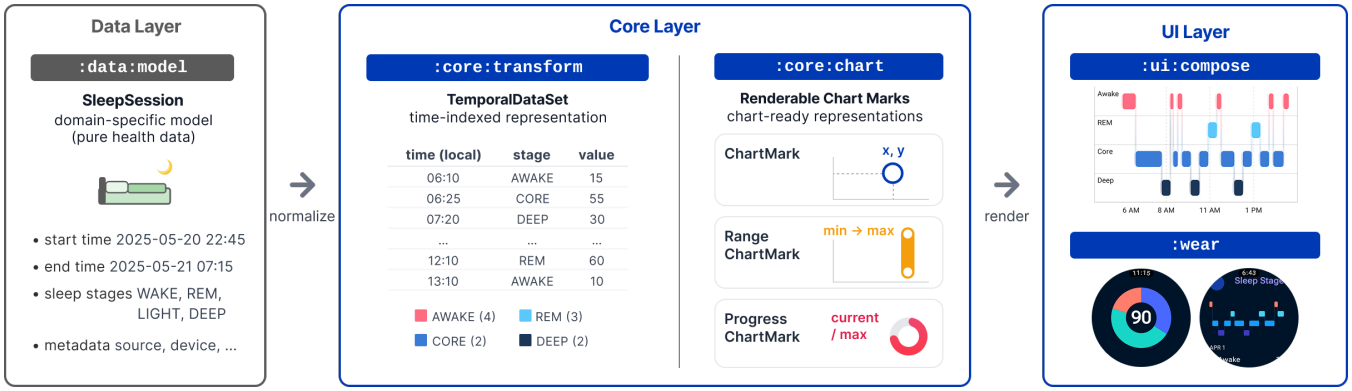

Figure 2: SalusChart system architecture and data flow illustrated using sample sleep data. A SleepSession is defined in the data layer (`:data:model`), normalized by `:core:transform` into a time-indexed `TemporalDataSet`, and converted by `:core:chart` into reusable chart marks. The resulting representation can then be rendered by presentation modules as a mobile sleep stage chart (`:ui:compose`) or Wear OS glance views (`:wear`).

Table 1: Chart types implemented in SalusChart, with example health metrics they display. A checkmark (✓) indicates that the chart type appears in each app (A: Apple Health, S: Samsung Health, G: Google Fit, F: Fitbit).

Chart Type	Health Data Examples	A	S	G	F
Line chart	Energy level, weight	✓	✓	✓	✓
Bar chart	Steps, active energy	✓	✓	✓	✓
Calendar view	Activity overview	✓	✓	✓	✓
Scatter plot	Blood pressure	✓		✓	✓
Activity ring	Move/Exercise/Stand goals	✓		✓	✓
Range bar chart	Heart rate, blood oxygen	✓	✓	✓	
Sleep stage chart	Awake/REM/Core/Deep	✓	✓		
Stacked bar chart	Sleep duration, nutrition		✓		✓
Progress bar	Daily step goal		✓	✓	
Range gauge	Blood glucose		✓		
Multi-segment gauge	Stress level, AGEs index		✓		
Pie chart	Nutrition		✓		
Ladder chart	Cardio fitness level	✓			

3.3 Architecture and Implementation

SalusChart realizes the four design goals described above through four high-level architectural decisions. First, Jetpack Compose, Android's declarative UI toolkit, lets developers describe *what* a chart should show, not *how* to draw it. This composition maps naturally to health dashboards, where cards typically contain summary information and a chart. In addition, it provides the foundation for reference line specifications (G3) and theme propagation via `CompositionLocal`, Compose's mechanism for implicitly passing data (e.g., theme) down through the UI tree (G4). Second, the library implements dual-level rendering (G2) as two separate composable families. Third, charts work out of the box with sensible defaults, such as colors from a pre-set theme, auto-scaled axes, and interaction enabled (G4), while advanced features, such as reference lines (G3) and paging (G1), are exposed as opt-in parameters. Finally, the library is split into independently publishable Maven artifacts (self-contained packages that developers add as dependencies), allowing developers to include only the modules they need.

SalusChart adapts along two axes: visual fidelity and data density. For visual fidelity, SalusChart achieves G2 through paired minimal and full chart representations. Minimal charts, such as `MinimalBarChart`, are used in dashboard cards, while full charts are used in detail views. Glance views for wearables are provided as separate components. The data density is controlled by two mutually exclusive parameters, `windowSize` for free scrolling and `pageSize` for paged navigation. For paged charts, horizontal paging is built on Compose Foundation's `HorizontalPager`, which separates horizontal swipes from the vertical scrolling of an enclosing dashboard without custom gesture handling. Chart interaction, such as bar or point selection, is enabled by default and can be disabled in embedded contexts.

Internally, SalusChart separates health-domain semantics and visual rendering. Application data begins as domain-specific health records, such as heart rate, blood pressure, sleep sessions, exercise, and step counts. These records differ in structure: some are single values, some are multi-property, and others describe ranges or durations. Rather than requiring each chart component to understand these domain-specific formats, SalusChart normalizes them into a shared time-indexed representation (`TemporalDataSet`) that can be reused across chart types. This representation stores timestamped values independently of any chart type, allowing temporal grouping, aggregation, labeling, and gap handling.

The normalized dataset is then converted to renderable chart marks. A single-value series is converted to simple x-y marks used by line, bar, scatter, and pie charts. Min-max aggregation is converted to range marks used by range-based charts, and progress-oriented data is represented as progress marks. Hence, `TemporalDataSet` and the chart-mark hierarchy (`ChartMark`, `RangeChartMark`, and `ProgressChartMark`) bridge heterogeneous health records and chart rendering.

The SalusChart library follows this separation across three layers shown in Figure 2. The `:data` layer defines health-domain models without charting dependencies. The `:core` layer contains reusable logic: `:core:transform` performs the data normalization and mark conversion described above, while `:core:chart` defines mark models, chart metrics, coordinate mapping, reference line specifications, and Canvas drawing utilities. The `:ui` layer exposes presentation modules: `:ui:compose` provides the Compose chart components, and `:ui:theme` propagates default chart colors and styles.

The same transformation and rendering core is reused across interfaces. The Wear OS extension reuses the shared data and core layers and adds only small-screen Compose components, so supporting it changes only the presentation layer.

4 Application Examples and Library Comparison

In this section, we evaluate SalusChart in two ways. To demonstrate its applicability, we first reproduce views from existing mHealth apps (Figure 1) and extend the SalusChart library to support Wear OS (Figure 3). We then compare its device coverage and implementation effort with those of general-purpose libraries.

Samsung Health and Apple Health. We reproduced dashboard and detail pages from Samsung Health and Apple Health as two standalone Android projects, each consuming SalusChart from Maven Central with a single dependency declaration. The Apple Health screens are reproduced in an Android application to show that SalusChart can render the same visual designs. They are not intended to imply that the library runs on iOS. The two examples in Figure 1 demonstrate different aspects of the library: the Samsung Health detail view uses a range bar chart with reference lines for the resting heart rate range (G3), while the Apple Health sleep view uses the sleep stage chart, a health-specific chart type that cannot be approximated from general primitives (G1).

SalusChart's APIs also scale from minimal to customized output: a chart can be built from a few required inputs with sensible defaults. Optional parameters then refine appearance, axis formatting, and in-chart annotations without restructuring the call (G4). We provide the full code for these examples on our supplemental website (`https://hdil-ys.github.io/SalusChart`).

Wear OS Extension. Unlike the first two examples (Figure 1), which consume SalusChart externally, the Wear OS extension adds a new `:wear` module (Figure 3). All four watch views reuse chart components from the Samsung Health and Apple Health examples.

Figure 3: Visualization examples on Wear OS smartwatches implemented using SalusChart's `:wear` module. From left to right: an activity ring showing daily goal progress, a heart rate display with daily min–max range, a multi-segment gauge indicating current stress level, and a sleep stage chart summarizing recent sleep.

Device Coverage. We verified rendering of eight views (Figures 1 and 3) on five emulator profiles: a compact phone (`Pixel 9`), a large phone (`Pixel 10 Pro XL`), a foldable (`Pixel 9 Pro Fold`), and two round Wear OS watches, all on API 36. Because SalusChart is built with Jetpack Compose, chart components are measured within their parent layout constraints and recompute their marks, axes, and labels for the available bounds. Across the three phone form factors, the same composables reflow to the available width without code change. All charts remained legible from the watch face to the foldable display. Screenshots for each profile are available on the supplemental website (`https://hdil-ys.github.io/SalusChart/emulator-screenshots`).

Implementation Effort. To provide initial evidence of reduced implementation effort, we reimplemented six charts from each of the Samsung Health and Apple Health examples. We implemented the 12 charts in SalusChart, MPAndroidChart, and Vico, matching each library's output to the original app's visual design (Table 2). SalusChart was the only library with a native chart type for all 12 charts. It also required the fewest lines of code and distinct API symbols (i.e., the charting-library classes and functions a developer must use) per chart. On average, SalusChart used about 20% fewer lines of code than either baseline and 24–58% fewer API symbols. Most of the difference arises in health-specific charts such as activity rings, sleep-stage charts, and range bars, where the general-purpose libraries lack a native type and fall back to emulation or hand-drawn Canvas. Per-chart and native-only breakdowns, screenshots, and the full procedure are available on the supplemental website (`https://hdil-ys.github.io/SalusChart/library-comparison`). Beyond implementation effort, only SalusChart provides a health data transformation layer and Wear OS support among the three (Table 2).

Table 2: Comparison with general-purpose Android chart libraries across 12 mHealth charts from our two sample apps.

Feature	SalusChart	MPAndroidChart	Vico
Native mHealth charts	**Yes**	No	No
Native coverage (of 12)	**12**	6	6
Health data transform	**Yes**	No	No
Compose-native	**Yes**	No	**Yes**
Wear OS support	**Yes**	No	No
Avg. lines of code	**28.6**	35.9	35.3
Avg. API symbols	**3.7**	4.9	8.8

5 Discussion and Future Work

SalusChart is a foundational step toward a dedicated data visualization library for mHealth apps, demonstrating the feasibility of reusable, health-specific visualization components. Building it also surfaced design trade-offs that we discuss below.

Tension between Flexibility and Complexity. The most prominent tension was a trade-off between flexibility and complexity: the more customization a library exposes, the larger its API surface and the steeper the learning curve. We addressed this challenge by providing sensible defaults for common cases while making advanced parameters opt-in. Our implementation effort comparison offers initial evidence of reduced development effort. A future study with Android developers could measure development time, API learnability, and subjective experience more directly.

Inclusivity and Accessibility. One major limitation with SalusChart is accessibility. Canvas-based rendering is essentially invisible to screen readers, and SalusChart currently does not expose chart content to TalkBack [4] or other assistive technologies. ChartA11y [23] and platform-level support, such as Swift Charts' Audio graphs [2], demonstrate how mobile charts can be made more accessible through semantic navigation, touch-based exploration, and audio feedback. Building on these, future work could expose SalusChart's chart marks and reference lines as accessible semantic and interaction targets, enabling screen readers to announce trends, values, and reference line crossings, and to expose individual data values when users select or navigate to them.

Toward a More Powerful mHealth Visualization Library. SalusChart currently runs only on Android, and developers need to retrieve data through platform health APIs, such as Health Connect on Android, and manually convert them into SalusChart's data types. A practical next step would be a companion adapter module that maps these APIs directly to `TemporalDataSet`. As Compose Multiplatform matures, it could allow SalusChart to run on iOS without requiring a complete rewrite of the codebase. The porting effort would likely focus on the UI layer, since `:data:model`, `:core:transform`, and `:core:util` have no Android-specific dependencies and would require little or no modification. With these extensions, SalusChart could grow into a more complete tool for mHealth app development, accelerating the creation of consistent, effective, and cross-platform health visualizations.

Acknowledgments

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korean Government (MSIT), Artificial Intelligence Graduate School Program, Yonsei University, under Grant RS-2020-II201361, and the Yonsei University Research Fund of 2026-22-0147.

References

[1] AnyChart for Android. https://github.com/AnyChart/AnyChart-Android. Accessed: 2026-04-25. 2

[2] Audio graphs — apple developer documentation. https://developer.apple.com/documentation/Accessibility/audio-graphs. Accessed: 2026-04-25. 4

[3] Charts. https://github.com/ChartsOrg/Charts. Accessed: 2026-04-25. 2

[4] Get started on android with talkback - android accessibility help. https://support.google.com/accessibility/android/answer/6283677?hl=en. Accessed: 2026-04-25. 4

[5] HelloCharts for Android. https://github.com/lecho/hellocharts-android. Accessed: 2026-04-25. 2

[6] MPAndroidChart. https://github.com/PhilJay/MPAndroidChart. Accessed: 2026-04-25. 2

[7] Swift Charts — Apple Developer Documentation. https://developer.apple.com/documentation/Charts. Accessed: 2026-04-25. 2

[8] Vico. https://github.com/patrykandpatrick/vico. Accessed: 2026-04-25. 2

[9] YCharts. https://github.com/codeandtheory/YCharts. Accessed: 2026-04-25. 2

[10] Y. A. Alshehhi, M. Abdelrazek, and A. Bonti. Analysis of Personal Data Visualisation Reviews on Mobile Health Apps. In *The Fifteenth International Conference on Advances in Computer-Human Interactions*, pp. 111–118, June 2022. 2

[11] Y. A. Alshehhi, M. Abdelrazek, B. J. Philip, and A. Bonti. Understanding User Perspectives on Data Visualization in mHealth Apps: A Survey Study. *IEEE Access*, 11:84200–84213, Aug. 2023. doi: 10.1109/ACCESS.2023.3302325 1, 2

[12] T. Blascheck, F. Bentley, E. K. Choe, T. Horak, and P. Isenberg. Characterizing Glanceable Visualizations: From Perception to Behavior Change. In *Mobile Data Visualization*, pp. 151–176. Chapman and Hall/CRC, Boca Raton, 1 ed., 2021. doi: 10.1201/9781003090823-5 1

[13] G. Chan, C. Nwagu, I. Odenigbo, A. Alslaity, and R. Orji. The Shape of Mobile Health: A Systematic Review of Health Visualization on Mobile Devices. *International Journal of Human–Computer Interaction*, 41(2):1154–1172, Jan. 2025. doi: 10.1080/10447318.2024.2313282 1, 2

[14] T. Horak, W. Aigner, M. Brehmer, A. Joshi, and C. Tominski. Responsive Visualization Design for Mobile Devices. In *Mobile Data Visualization*, pp. 33–66. Chapman and Hall/CRC, Boca Raton, 1 ed., 2021. doi: 10.1201/9781003090823-2 2

[15] A. Jena, M. Butler, T. Dwyer, K. Ellis, U. Engelke, R. Kirkham, K. Marriott, C. Paris, and V. Rajamanickam. The Next Billion Users of Visualization. *IEEE Computer Graphics and Applications*, 41(2):8–16, Mar. 2021. doi: 10.1109/MCG.2020.3044071 2

[16] S.-H. Kim. A Systematic Review on Visualizations for Self-Generated Health Data for Daily Activities. *International Journal of Environmental Research and Public Health*, 19(18):11166, Sept. 2022. doi: 10.3390/ijerph191811166 2

[17] Y.-H. Kim, B. Lee, A. Srinivasan, and E. K. Choe. Data@Hand: Fostering Visual Exploration of Personal Data on Smartphones Leveraging Speech and Touch Interaction. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pp. 1–17. Association for Computing Machinery, New York, NY, USA, May 2021. doi: 10.1145/3411764.3445421 2

[18] B. Lee, S. Bae, M. Li, and E. K. Choe. Envisioning Mobile Data Visualization Libraries for Digital Health. arXiv, Apr. 2026. doi: 10.48550/arXiv.2604.24448 1, 2

[19] B. Lee, M. Brehmer, P. Isenberg, E. K. Choe, R. Langner, and R. Dachselt. Data Visualization on Mobile Devices. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI EA '18, pp. 1–8. Association for Computing Machinery, New York, NY, USA, Apr. 2018. doi: 10.1145/3170427.3170631 1

[20] B. Lee, E. K. Choe, P. Isenberg, K. Marriott, and J. Stasko. Reaching Broader Audiences with Data Visualization. *IEEE Computer Graphics and Applications*, 40(2):82–90, Mar. 2020. doi: 10.1109/MCG.2020.2968244 2

[21] B. Lee, R. Dachselt, P. Isenberg, and E. K. Choe. *Mobile Data Visualization*. CRC Press, 2021. doi: 10.1201/9781003090823 2

[22] J. Meyer, A. Kazakova, M. Büsing, and S. Boll. Visualization of Complex Health Data on Mobile Devices. In *Proceedings of the 2016 ACM Workshop on Multimedia for Personal Health and Health Care*, MMHealth '16, pp. 31–34. Association for Computing Machinery, New York, NY, USA, Oct. 2016. doi: 10.1145/2985766.2985774 1, 2

[23] Z. Zhang, J. R. Thompson, A. Shah, M. Agrawal, A. Sarikaya, J. O. Wobbrock, E. Cutrell, and B. Lee. ChartA11y: Designing accessible touch experiences of visualizations with blind smartphone users. In *Proceedings of the 26th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '24. Association for Computing Machinery, New York, NY, USA, 2024. doi: 10.1145/3663548.3675611 4