Altify: Shift-left Icon Alt-text for Mobile App Developers

Sabrina Haque
Computer Science and Engineering Department
University of Texas at Arlington
Arlington, Texas, USA
sxh3912@mavs.uta.edu

Christoph Csallner
Computer Science and Engineering Department
University of Texas at Arlington
Arlington, Texas, USA
csallner@uta.edu

Abstract

Meaningful descriptions or alt-texts are essential for mobile app accessibility, yet it is often missing or uninformative, particularly for UI icons. Existing approaches typically require large task-specific training sets and generate or audit alt-text post-development, after an interface is complete. We report findings from a formative study showing that developers strongly prefer alt-text support at the moment they add UI elements, rather than after screen completion. Based on this insight, we propose Altify, a system that automates alt-text generation for UI icons during app development (i.e., using only partial screen context). Altify extracts relevant UI information from the view hierarchy tree and icon image to generate short, descriptive alt-text suggestions. An initial evaluation on a benchmark dataset with automated metrics suggests that meaningful alt-text can be generated even without full-screen context.

CCS Concepts

• **Human-centered computing → Accessibility systems and tools**; • **Software and its engineering → Software usability**.

Keywords

Developer support, visual impairment, icon context, icon alt-text

ACM Reference Format:
Sabrina Haque and Christoph Csallner. 2026. Altify: Shift-left Icon Alt-text for Mobile App Developers. In *Web4All 2026 23rd International Web for All Conference (W4A '26), April 13–14, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3800424.3800427

1 Introduction

Automatic generation of alt-text for an app icon currently either yields low-quality alt-text or requires programmers to wait until the code of a complete app screen is available. Waiting until a screen is complete causes programmers to lose important mental context information (which makes it harder for the developer to judge inferred alt-text). Taken together, most programmers today do not use tools for generating icon alt-text (and we have also observed this lack of tool use in our surveys).

Across web and mobile apps, user interfaces rely on small interactive graphical UI elements (i.e., icons and image buttons), whose meaning is primarily conveyed visually. To support non-visual interaction, platforms encourage developers to provide for these UI elements short textual descriptions, aka alt-text (e.g., `alt` or ARIA tags on the web and `contentDescription` on Android). Most users with visual impairments rely on such alt-text (as conveyed via screen readers such as TalkBack and VoiceOver [19, 21, 38]) to learn about an UI element's purpose and navigate the app [10].

Although guidelines and regulations mandate digital accessibility [1, 36], adding alt-text is often an afterthought in development, e.g., due to the required manual effort [12], tight deadlines, lack of prioritization during development [15], and an overall lack of immediate impact on core functionality [6, 16]. So many apps ship with inadequate or missing alt-text [3, 12, 16]. Missing alt-text remains common in Android apps, with 86% of clickable ImageViews and 55% of ImageButtons lacking a contentDescription in Rico [11, 33] and similar rates persisting in the 2024 MUD dataset [14].

In practice, most accessibility tools focus on post-development activities. Addressing accessibility issues late in development requires developers to revisit completed work, which disrupts workflow and introduces context-switching overhead [8, 12]. Prior work therefore emphasizes *shift-left accessibility*, calling for accessibility tools that support accessibility earlier and more continuously during development [6, 13, 28]. A prior study on icon labeling has also called for tool support at the moment of change "At a smaller scale, developer tools might also target moments of change corresponding to new interface elements [..]" [16]. Despite these calls, it remains unclear when developers prefer to receive automated alt-text support during UI development.

Recent work has made great progress toward automatically inferring alt-text for UI elements. While early deep-learning techniques (LabelDroid [9] and Coala [27]) struggled with generating meaningful alt-text for less-common icons, more recent vision-transformers (Pix2Struct [22]) and vision-language models such as PaliGemma [7] achieve strong performance but typically assume access to a complete UI screenshot. Overall these approaches also rely on large task-specific training datasets. As a result, these approaches can generate meaningful alt-text only after an entire screen is implemented, when developers may have already lost important mental context needed to judge the output.

To address these gaps, we divide our contributions into two parts. First, we conduct a formative study to understand developers' current practices and preferences regarding automated alt-text support, with a focus on when such support would be most useful. Second, motivated by these findings, we propose Altify for generating icon alt-text during development (i.e., using partial UI context) and present an initial feasibility evaluation.

ACM ISBN 979-8-4007-2372-8/2026/04
https://doi.org/10.1145/3800424.3800427

While we instantiate Altify on Android's XML-based UI framework, the same design principles apply to newer Android frameworks such as Jetpack Compose and other platforms with comparable UI structures, such as iOS and the web. Our Altify implementation, data, fine-tuning configuration, and evaluation scripts are publicly available [20]. To summarize, we study the following two research questions (RQs).

RQ1: When do developers (and especially junior ones) want automated alt-text support during UI development?

RQ2: Can we generate high-quality alt-text for UI elements when the developer adds or edits a UI element, i.e., based on the current (partial) screen code with minimum task-specific prior data?

2 RQ1: Developers: "Shift-left Alt-text"

As existing work has not yet asked developers when in the development process they would prefer to generate alt-text [25, 37], we conducted an online survey. We primarily targeted junior developers, as they are less familiar with accessibility requirements and tend to produce lower-quality alt-texts [9]. We recruited participants through multiple channels, including computer science students with app development experience, Reddit, and LinkedIn outreach. To encourage honest feedback the survey was anonymous.

We received 52 responses, skewed toward junior developers (i.e., 60% < 1 year experience in mobile app development, 19% 1–3 years, 6% 4–6 years, and 15% over 6 years). As their primary development platform, 60% selected Android, followed by iOS (13%), Android+iOS (13%), web (10%), and none (4%). A minority (23%) had worked on apps that required accessibility support (*"Have you worked on apps that require accessibility features (e.g., support for screen readers)?"*).

68% indicated that they currently add alt-text to at least some image-based UI element (*"Do you currently add alt-texts for image-based UI elements (e.g. icons, buttons)?"*). When asked how they generate alt-text, the most common technique was manual authoring (50%), followed by other (12%) (*"How do you currently generate alt-text for image-based UI elements (e.g., icons, buttons)?"*). The latter includes answers such as using predefined text, AI-generated suggestions with manual review, and relying on third-party teams.

Table 1: Developer interest and preferred project stage for automated alt-text generation. Omits "no response": 6%, 4%.

Would use IDE plugin to generate alt-text	**(%)**
Yes, regularly	23
Yes, occasionally	46
Might use if certain criteria met	15
No	10
When to generate alt-text	**(%)**
Wireframing / initial design	19
As soon as I add a UI element to the screen code	44
After I added all UI elements to the current screen area	17
After we added all UI elements to the current screen	4
After we added all UI elements to all app screens	12

The majority (69%) indicated they would use automated alt-text generation occasionally or regularly, with another 15% expressing conditional interest (Table 1: *"Would you be interested in using a plugin for automated generation of alt-texts in your development workflow?"*). Conditions cited include minimal disruption to development flow, ease of integration, and specific features such as *"It would have to identify any text in the image and take that into account when generating alt text"*.

Only a small minority (16%) prefers to delay using such a tool at least until an entire screen is available (Table 1: *"If such a tool was available, at which stages of development would you find it most helpful?"*). The most popular option (44%) was ***"Screen UI code in-progress (as soon as I add a UI element to the screen code)"***. Most participants (79%) rated the ability to edit or approve generated alt-text as important or extremely important (*"How important is it for the tool to allow customization of the generated alt-text (e.g., editing or approving suggestions)?"*). A majority (56%) preferred to review generated alt-text immediately after adding the UI element (*"If you would like to review the generated alt-text to ensure accuracy, when would you prefer to do it?"*), when relevant context is still fresh and evaluation requires less cognitive effort.

3 Altify Design

Figure 1 illustrates Altify's workflow. Based on our formative study, Altify generates a UI icon's alt-text when a developer adds or edits an icon in the current screen's view hierarchy code. To leverage the context available (the icon's pixels and textual info), Altify uses a multimodal fine-tuned LLM (currently GPT-4o).

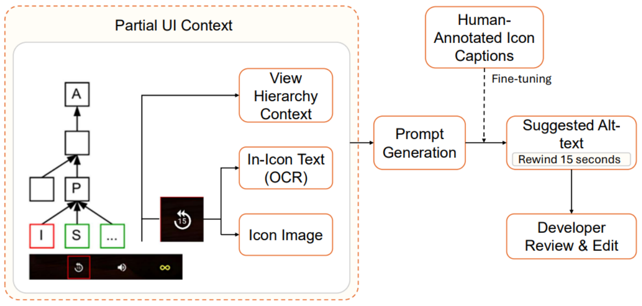


Figure 1: On icon add or edit in the view hierarchy, Altify extracts the current context of icon (I), parent (P), siblings (S), and activity (A), the icon image, and in-icon text.

Altify detects a new or edited icon based on UI class (currently ImageButton and ImageView) and extracts its relevant *"developer-side"* properties[1] from the view hierarchy, i.e., the app screen ("activity") name and the icon's textual properties (i.e., class name, resource ID, and on-screen text).

As related UI elements are typically grouped within a UI container, Altify also identifies the icon's parent node in the view hierarchy and extracts similar textual properties from this parent and the parent's current direct children—the icon's sibling nodes. Listing 1 shows the icon context extracted for the Figure 1 icon.

Icons sometimes contain embedded text that conveys functional meaning but is not represented in the view hierarchy. We extract

[1]We call these properties "developer-side" as developers interact with and thus benefit from them frequently (e.g., during coding and debugging), leading to higher quality than accessibility attributes that a developer typically interacts with rarely.

```
"app_activity_name": "is.abide.ui.PlayerActivity",
"UI_element_info": {
    "class_name": "AppCompatImageButton",
    "resource_id": "rewind_button" },
"parent_node": [
    { "resource_id": "toggle_layout" },
    { "class": "LinearLayout" } ],
"sibling_nodes": [
    { "resource_id": "background_music_button",
      "class": "AppCompatImageButton" },
    { "resource_id": "autoplay_button",
      "class": "ToggleButton" } ]
```

Listing 1: The Figure 1 icon's context (icon, parent, siblings, and activity) that Altify extracts from the view hieararchy.

such in-icon text using optical character recognition (OCR) with EasyOCR [17] and add it to the icon's context. While multimodal LLMs can recognize text, OCR is cheap, reliable, and local. Altify uses the extracted context in the Listing 2 prompt template.

```
"You are an accessibility assistant to a mobile app developer.
The attached image is of a UI element (icon) which has the
following view hierarchy content:
{icon context}
Generate a short (within 2-7 words), DESCRIPTIVE alt-text for the
UI element. Provide only the alt-text as output, nothing else.
Describe the element as if you were the app developer to HELP
VISION-IMPAIRED USERS understand its FUNCTIONALITY and PURPOSE.
Avoid generic words like 'button', 'image', 'icon' etc."
```

Listing 2: Prompt template of Altify.

Altify's output is a suggestion that developers can accept, revise, or regenerate as the UI evolves. This aligns with developers' preference for human-in-the-loop workflows and helps prevent incorrect or misleading descriptions from being applied automatically.

4 RQ2: Initial Quality Evaluation

As an initial quality evaluation, we compare generated alt-texts against human-written descriptions using automated text-similarity metrics [18]. While such metrics do not directly measure accessibility effectiveness, they provide an initial signal of whether generated alt-text aligns with human-perceived icon meaning.

As ground truth we use the widely used Widget Captioning dataset (WC20) [23]. WC20 extends the Rico [11] dataset of Android UI screenshots and view hierarchies by adding to each UI element up to three independent human annotations. Following prior work [27] we extract from 8,201 WC20 app screens 21,314 icon elements (ImageView and ImageButton).

We compare Altify against representative approaches across three categories: classic deep-learning (LabelDroid [9] & Coala [27]), vision-transformers (Pix2Struct [22]), and vision-language models (PaliGemma [7]). We evaluate all methods on the standard WC20 test split, which contains 1,635 icons and their 4,419 human-written (ground truth) labels. We measure alignment with these labels with standard text distance metrics from MS COCO [18], where higher scores mean closer matches. Among the metrics, CIDEr and SPICE have shown stronger alignment with human judgments [4].

To fine-tune LabelDroid[2] and Coala[3] on WC20 we use the tool authors' released code and configurations. For Pix2Struct[4] and PaliGemma[5], we use publicly available models already fine-tuned on WC20 and evaluate them under two visual-context conditions. In full-screen (their original setup), we provide the complete UI screenshot as input and mark the target icon with a red bounding box. In partial-screen, we restrict the visual input to the icon's local context, retaining the icon, its parent container, and sibling elements, leaving the rest of the screen blank (black). The latter setup approximates development-time scenarios where the app screen is incomplete.

We fine-tune Altify on a small, diverse subset of WC20. Following prior work [24], we sample 15 representative icons from each of 99 icon-concept clusters, plus a catch-all "other" category, resulting in 1,425 icons drawn exclusively from the WC20 training split. No apps or screens overlap with the test set. We fine-tune Altify using gpt-4o-2024-08-06 via the OpenAI API [31, 32] for three epochs under identical conditions. This setup is conservative in the sense that we limited Altify to train on a small subset, i.e., 8% of the WC20 training icons (1,425 / 18,176), while training the other tools on WC20's full 18,176 icons.

4.1 Altify: High Scores on Automated Metrics

In our Table 2 results the classic (fine-tuned) deep-learning models achieved similar CIDEr scores as our zero-shot Altify baseline (i.e., Altify with all its input except the fine-tuning step). This observation suggests that modern LLMs already capture substantial visual semantics for icon description without task-specific adaptation. The fine-tuned Altify substantially improves over its zero-shot baseline, indicating that even limited task-specific fine-tuning can meaningfully enhance the model's ability to generate semantically plausible icon alt-text from partial UI context.

Table 2: Automated quality metric scores of Altify vs. LabelDroid & Coala baselines, Pix2Struct vision-transformer, and PaliGemma VLM for full/partial (f/p) UI; M = METEOR; bold = best score; underline = runner-up; higher scores align closer with human annotations.

	BLEU-1	BLEU-2	ROUGE	M	CIDEr	SPICE
LabelDroid	28.4	17.7	31.7	12.5	59.4	10.6
Coala	35.3	24.1	34.0	14.2	66.3	9.4
Pix2Struct-large$_f$	41.9	28.1	41.7	18.4	89.5	14.8
Pix2Struct-large$_p$	35.6	23.9	35.1	15.7	73.7	10.7
PaliGemma-448$_f$	<u>55.4</u>	<u>41.6</u>	<u>56.2</u>	<u>24.7</u>	<u>127.1</u>	<u>21.8</u>
PaliGemma-448$_p$	53.6	39.3	55.9	24.1	123.5	20.6
Altify (0-shot)	34.1	17.4	37.4	18.7	61.6	11.9
Altify	**61.0**	**44.5**	**59.2**	**27.7**	**138.3**	**23.2**

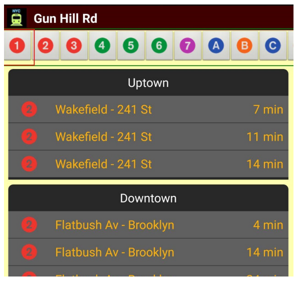
Altify consistently performs better than all existing approaches, particularly in partial screen conditions ("p" variants). Across existing methods, performance generally degrades as available visual context is reduced from full to partial screens (which is expected as it removes context information). Overall, these results suggest that generating meaningful icon alt-text during development (i.e.,

[2]https://github.com/chenjshnn/LabelDroid, accessed Jan. 2026
[3]https://github.com/fmehralian/COALA, accessed Jan. 2026
[4]https://huggingface.co/google/pix2struct-widget-captioning-large, accessed Jan. 2026
[5]https://huggingface.co/google/paligemma-3b-ft-widgetcap-448, accessed Jan. 2026

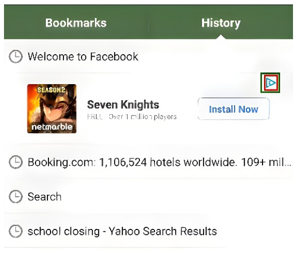
from partial screen context) is feasible and Altify is better suited for the task than existing approaches.

4.2 Human Analysis of Low-Scoring Alt-texts

To better understand the cases where Altify scored low on automated metrics, we manually reviewed 50 random test icons from the 200 for which Altify's outputs had the lowest overlap with ground truth labels. Our goal was not to reassess metric performance, but to understand why mismatches occur. Following are the main causes (Figure 2 shows two representative examples).


Gun Hill Rd
1 2 3 4 5 6 7 A B C
Uptown
Wakefield - 241 St 7 min
Wakefield - 241 St 11 min
Wakefield - 241 St 14 min
Downtown
Flatbush Av - Brooklyn 4 min
Flatbush Av - Brooklyn 14 min

(a) ***NYC Subway Time***
Ref: "first page", "open selected window";
$\text{PaliGemma}_\text{P}$: "select number";
Altify: "select line 1".


Bookmarks
History
Welcome to Facebook
Seven Knights
Install Now
Booking.com: 1,106,524 hotels worldwide. 109+ mil...
Search
school closing - Yahoo Search Results

(b) ***Boat Browser Mini***
Ref: "advertizer description", "click on advertisement", "this is an advertisement";
$\text{PaliGemma}_\text{P}$: "play video";
Altify: "play button".

Figure 2: Sample screens (excerpts for space) where Altify deviates from the ground truth: Target icon (red box); icon's parent (green box, not marked for tools); ground truth (Ref).

Context-driven Semantic Mismatch (32/50): The view hierarchy's UI context strongly influences the generated alt-text. In the Figure 2a example, the substring "line" occurs in the icon's context several times, yielding a Altify alt-text that deviates from the reference labels, while still remaining meaningful. These cases often reflect semantically valid but differently phrased outputs.

Icon Image Misleads Inference (11/50): The icon's visual appearance leads to ambiguous interpretations. For example, the Figure 2b icon resembles both a stylized Google Play Store icon and the widely used media "play" icon. While contextual cues suggest an ad, Altify's output does not fully capture the intended function.

Vague Icon (7/50): The icon's ground truth labels differ widely among each other, indicating annotator confusion. Developer review is especially valuable here, as developers best understand the intended meaning.

Overall, low metric scores reflected ambiguity rather than unusable alt-text. This reinforces the importance of allowing developers to review and edit generated alt-text during development.

4.3 Threats to Validity

Internal Validity: Altify relies on the context extracted from UI view hierarchies and screen images. As detailed in our error analysis (Section 4.2), some failures stem from misleading or noisy DOM content, ambiguous icon images, or vague semantics that confuse even human annotators. Additionally, errors in DOM parsing or OCR (e.g., missing resource IDs or failed text extraction) may degrade input quality and affect the quality of generated text.

External Validity: The WC20 dataset may have leaked into GPT-4o's training data, which may skew RQ2 results. While this risk is hard to quantify, Altify's 0-shot performance (Table 2) is notably weaker than Altify's (which fine-tunes on our WC20 sample), suggesting GPT-4o likely did not memorize the WC20 test data.

Construct Validity: Metrics like CIDEr do not always align with human judgment, especially in subjective, context-rich UI tasks. To mitigate this threat, we will conduct a human subject study.

5 Related Work

Most existing accessibility tools operate post-development, e.g., recent work on using (LLMs) to detect and repair accessibility issues in Android apps [2]. FixAlly suggests automated repairs [26], but requires developers to return to already implemented UI components. AltAuthor and AltCat generate alt-text for web images using broad webpage-level context, targeting all on-page images (vs. UI icons) [34, 35]. AltCat further emphasizes culturally appropriate descriptions based on the target country. In contrast, our work targets UI icons and supports development-time inference (i.e., relying only on partial UI context).

AI coding assistants (e.g., GitHub Copilot) may improve accessibility, but typically only when developers explicitly prompt for accessibility-aware output, making their effectiveness dependent on developer expertise and awareness [29]. Extensions such as CodeA11y add accessibility-focused guidance by prompting developers to address errors or complete placeholders [30]. While such tools improve compliance and awareness, they still rely on developers to manually author alt-text.

Many tools also require developers to manually locate and fix problems, which is time-consuming [26]. Static analysis tools such as Android Lint [5] flag missing alt-text but do not generate meaningful suggestions, and developers often disable such warnings [37].

6 Conclusions And Future Work

Alt-text is essential for accessibility, yet app UI icons often lack meaningful descriptions, limiting accessibility for screen reader users. Existing approaches require extensive labeled datasets, assume full UI screens, and operate post-development. Through a formative study, we show that developers prefer receiving alt-text support early. We thus introduced Altify, a development-time approach for generating icon alt-text from partial UI context. Our initial evaluation suggests that semantically meaningful alt-text can be generated even when full screen context is unavailable, supporting the feasibility of early, in-context accessibility assistance.

We plan to integrate Altify into real development workflows and evaluate its impact on usability, trust, and accessibility outcomes. Altify could be integrated into IDEs using existing mechanisms such as inline hints or quick-fix recommendations.

Acknowledgments

Christoph Csallner has a potential research conflict of interest due to a financial interest with Microsoft and The Trade Desk. A management plan has been created to preserve objectivity in research in accordance with UTA policy.

References

[1] ADA.gov. 2023. The Americans with Disabilities Act (ADA). https://www.ada.gov/.

[2] Wajdi Aljedaani, Ahmed Aljohani, Marcelo M Eler, Abdulrahman Habib, and Hyunsook Do. 2025. LLMs for Accessibility in Mobile Apps: Detection and Repair. In *Proceedings of the 22nd International Web for All Conference*. 172–183.

[3] Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek. 2020. Accessibility issues in Android apps: State of affairs, sentiments, and ways forward. In *Proc. ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*. 1323–1334.

[4] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2016. SPICE: Semantic Propositional Image Caption Evaluation. In *Proc. 14th European Conference on Computer Vision (ECCV)*. Springer, 382–398.

[5] Android Studio. 2025. Improve your code with lint checks. https://developer.android.com/studio/write/lint.

[6] Aleksander Bai, Heidi Camilla Mork, and Viktoria Stray. 2017. A cost-benefit analysis of accessibility testing in agile software development: Results from a multiple case study. *International Journal on Advances in Software* 10, 1&2 (2017), 96–107.

[7] Lucas Beyer et al. 2024. PaliGemma: A versatile 3B VLM for transfer. *arXiv preprint arXiv:2407.07726* (2024).

[8] Tingting Bi, Xin Xia, David Lo, John Grundy, Thomas Zimmermann, and Denae Ford. 2022. Accessibility in software practice: A practitioner's perspective. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 4 (2022), 1–26.

[9] Jieshan Chen et al. 2020. Unblind Your Apps: Predicting Natural-Language Labels for Mobile GUI Components by Deep Learning. In *Proc. ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*. ACM, 322–334.

[10] Sen Chen, Chunyang Chen, Lingling Fan, Mingming Fan, Xian Zhan, and Yang Liu. 2021. Accessible or not? An empirical investigation of Android app accessibility. *IEEE Transactions on Software Engineering* 48, 10 (2021), 3954–3968.

[11] Biplab Deka et al. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proc. 30th annual ACM symposium on User Interface Software and Technology (UIST)*. 845–854.

[12] Marianna Di Gregorio, Dario Di Nucci, Fabio Palomba, and Giuliana Vitiello. 2022. The making of accessible Android applications: An empirical study on the state of the practice. *Empirical Software Engineering* 27, 6 (2022), 145.

[13] DigitalA11Y. 2025. Shift Left Accessibility in Design, Development and Testing. https://www.digitala11y.com/connecting-dots-of-an-accessibility-audit/.

[14] Sidong Feng, Suyu Ma, Han Wang, David Kong, and Chunyang Chen. 2024. Mud: Towards a large-scale and noise-filtered UI dataset for modern style UI modeling. In *Proc. 2024 CHI Conference on Human Factors in Computing Systems*. 1–14.

[15] Sidong Feng, Suyu Ma, Jinzhong Yu, Chunyang Chen, Tingting Zhou, and Yankun Zhen. 2021. Auto-Icon: An automated code generation tool for icon designs assisting in UI development. In *Proc. 26th International Conference on Intelligent User Interfaces (IUI)*. 59–69.

[16] Raymond Fok, Mingyuan Zhong, Anne Spencer Ross, James Fogarty, and Jacob O Wobbrock. 2022. A Large-Scale Longitudinal Analysis of Missing Label Accessibility Failures in Android Apps. In *Proc. 2022 CHI Conference on Human Factors in Computing Systems*. 1–16.

[17] GitHub. 2023. EasyOCR. https://github.com/JaidedAI/EasyOCR.

[18] GitHub. 2023. Microsoft COCO Caption Evaluation. https://github.com/tylin/coco-caption.

[19] Google. 2024. Get started on Android with TalkBack. https://support.google.com/accessibility/android/answer/6283677.

[20] Sabrina Haque. 2025. Replication package: Code and data for this paper. https://figshare.com/s/6452b23b9589f747c540.

[21] iOS. 2024. VoiceOver on iPhone. https://support.apple.com/guide/iphone/turn-on-and-practice-voiceover-iph3e2e415f/ios.

[22] Kenton Lee et al. 2023. Pix2Struct: Screenshot parsing as pretraining for visual language understanding. In *Proc. International Conference on Machine Learning (ICML)*. PMLR, 18893–18912.

[23] Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020. Widget captioning: Generating natural language description for mobile user interface elements. *arXiv preprint arXiv:2010.04295* (2020).

[24] Thomas F Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. Learning design semantics for mobile apps. In *Proc. 31st Annual ACM Symposium on User Interface Software and Technology*. 569–579.

[25] Forough Mehralian, Titus Barik, Jeff Nichols, and Amanda Swearngin. 2024. Automated Code Fix Suggestions for Accessibility Issues in Mobile Apps. *arXiv preprint arXiv:2408.03827* (2024).

[26] Forough Mehralian, Ziyao He, and Sam Malek. 2024. Automated Accessibility Analysis of Dynamic Content Changes on Mobile Apps. In *Proc. IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE, 482–494.

[27] Forough Mehralian, Navid Salehnamadi, and Sam Malek. 2021. Data-driven accessibility repair revisited: On the effectiveness of generating labels for icons in Android apps. In *Proc. 29th Symposium on the Foundations of Software Engineering (FSE)*. ACM, 107–118.

[28] Darliane Miranda and João Araujo. 2022. Studying industry practices of accessibility requirements in agile development. In *Proc. 37th ACM/SIGAPP Symposium on Applied Computing*. 1309–1317.

[29] Peya Mowar, Yi-Hao Peng, Aaron Steinfeld, and Jeffrey P Bigham. 2024. Tab to autocomplete: The effects of AI coding assistants on web accessibility. In *Proc. 26th International ACM SIGACCESS Conference on Computers and Accessibility*. 1–6.

[30] Peya Mowar, Yi-Hao Peng, Jason Wu, Aaron Steinfeld, and Jeffrey P Bigham. 2025. CodeA11y: Making AI Coding Assistants Useful for Accessible Web Development. *arXiv preprint arXiv:2502.10884* (2025).

[31] OpenAI. 2024. Fine-Tuning with OpenAI Models. https://platform.openai.com/docs/guides/fine-tuning. Accessed February 1, 2025..

[32] OpenAI. 2024. OpenAI developer platform. https://platform.openai.com/docs/overview. Accessed February 1, 2025..

[33] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O Wobbrock. 2018. Examining image-based button labeling for accessibility in Android apps through large-scale analysis. In *Proc. 20th International ACM SIGACCESS Conference on Computers and Accessibility*. 119–130.

[34] Hyungwoo Song, Jaehoon Choi, Minjeong Shin, Bongwon Suh, and Hyunggu Jung. 2025. AltCAT: An Alt Text Authoring Tool with Automatic Generation and Culturally-Aware Translation. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility*. 1–5.

[35] Hyungwoo Song, Minjeong Shin, Yeonjoon Kim, Kyochul Jang, Jaehoon Choi, Hyunggu Jung, and Bongwon Suh. 2025. AltAuthor: A Context-Aware Alt Text Authoring Tool with Image Classification and LMM-Powered Accessibility Compliance. In *Companion Proceedings of the 30th International Conference on Intelligent User Interfaces*. 124–128.

[36] U.S. Access Board. 2018. About the ICT Accessibility 508 Standards and 255 Guidelines. https://www.access-board.gov/ict/.

[37] Christopher Vendome, Diana Solano, Santiago Liñán, and Mario Linares-Vásquez. 2019. Can everyone use my app? An empirical study on accessibility in Android apps. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 41–52.

[38] WebAIM. 2024. Screen Reader User Survey. https://webaim.org/projects/screenreadersurvey10/.