What You See Is Not What You Tap: Detecting Misalignments between Visual and Interactive Boundaries in Mobile Apps

Qingyang Qiu
College of Computer Science and Artificial Intelligence
Fudan University
Shanghai, China
qyqiu25@m.fudan.edu.cn

Xuan Wang
College of Computer Science and Artificial Intelligence
Fudan University
Shanghai, China
xuanwang25@m.fudan.edu.cn

Jintao Wang
College of Computer Science and Artificial Intelligence
Fudan University
Shanghai, China
jintaowang24@m.fudan.edu.cn

Yongxiang Hu
College of Computer Science and Artificial Intelligence
Fudan University
Shanghai, China
yongxianghu23@m.fudan.edu.cn

Hailiang Jin
Meituan
Shanghai, China
jinhailiang@meituan.com

Xingjian Yang
College of Computer Science and Artificial Intelligence
Fudan University
Shanghai, China
yangxj24@m.fudan.edu.cn

Yao Xu
College of Computer Science and Artificial Intelligence
Fudan University
Shanghai, China
yaoxu24@m.fudan.edu.cn

Shiyu Guo
Meituan
Shanghai, China
guoshiyu03@meituan.com

Juxing Yuan
Meituan
Beijing, China
yuanjuxing@meituan.com

Yangfan Zhou
College of Computer Science and Artificial Intelligence
Fudan University
Shanghai, China
zyf@fudan.edu.cn

Abstract

Predictability is a central principle of mobile app interaction design, requiring that an app's responses align with user expectations. Beyond the well-tested functional misalignment, our industrial practice highlights a frequently violated yet formally unexplored aspect: *visual-interactive misalignment* (VIM), a discrepancy between an element's visual and interactive boundaries. Such misalignments can cause user taps to be ignored or trigger unintended actions, and consequently degrade the overall experience. Due to their dynamic nature during runtime, existing static analysis tools are inherently incapable of detecting these defects. To address this critical gap, we present `TouchAlign`, a black-box tool that detects VIMs. `TouchAlign` works by comparing an element's visual boundary, extracted via object detection, against its interactive boundary, which is mapped out by simulating user taps around the element. Integrated into the software quality assurance pipeline of Meituan App (an on-demand lifestyle platform with over 700 million users), `TouchAlign` has effectively uncovered critical defects that had eluded conventional testing, thereby helping preserve the predictability of mobile applications.

CCS Concepts

• **Software and its engineering** → **Software testing and debugging**; • **Human-centered computing** → *User interface programming*.

Keywords

GUI Testing, Visual-Interactive Misalignment, Mobile Application, User Experience

ACM Reference Format:
Qingyang Qiu, Xuan Wang, Jintao Wang, Yongxiang Hu, Hailiang Jin, Xingjian Yang, Yao Xu, Shiyu Guo, Juxing Yuan, and Yangfan Zhou. 2026. What You See Is Not What You Tap: Detecting Misalignments between Visual and Interactive Boundaries in Mobile Apps. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 5–9, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3803437.3805258

FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/26/07
https://doi.org/10.1145/3803437.3805258

1 INTRODUCTION

Predictability serves as a fundamental principle of mobile app design [36, 40], emphasizing the Graphical User Interface (GUI) behaves as users expect. For example, tapping a "trash can" icon should intuitively trigger a delete operation. Therefore, to ensure a positive user experience, substantial academia researches are proposed to verify the function of interactive GUI elements consistent with their visual representation [19, 28].

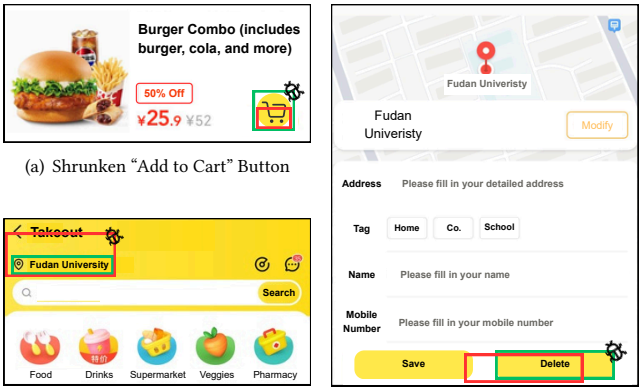
However, aside from functional inconsistencies, our industrial practice shows that *misplaced interactive areas* can also harm *Predictability*. In particular, we observe recurring user complaints in which intended actions fail to trigger when users tap visible elements, or unintended operations are activated by touching areas that appear inactive. We identify the root cause as a subtle yet pervasive class of defects, which we term *visual-interactive misalignment (VIM)*: a discrepancy between an element's visual boundary and its underlying interactive boundary.

Although it may appear counterintuitive, this misalignment is indeed an unintended consequence of engineering practices meant to improve usability [21, 44]. To meet design guidelines for minimum touch target sizes and the required spacing between adjacent targets [14, 23, 34], developers often manually adjust the interactive areas of elements. This practice, while well-intentioned, inevitably decouples the user-visible GUI elements from the underlying interactive regions. Consequently, such manual and often heuristic adjustments can easily introduce discrepancies between what users see and where they can interact.

VIMs can directly lead to interaction failures or unexpected behavior. As illustrated in Figure 1, an element's interactive boundary may be *shrunken*, *bloated*, or *offset* relative to its visual boundary. Based on our assurance experience at Meituan, such defects can have tangible consequences: a shrunken "Add to Cart" button may prevent users from completing purchases, while an unintended activation of an offset "Delete" button can cause irreversible data loss and undermine user trust. Moreover, we also conducted an empirical study on 16 popular apps to investigate the prevalence of VIMs. Our manual inspection of 134 of their core screens revealed 53 distinct VIMs, indicating that VIM is a widespread and largely unaddressed issue in the mobile industry. Detecting these defects before release is therefore critical for app quality and user experience (UX).

To the best of our knowledge, no existing tool is capable of detecting VIM. The most related tools are accessibility checkers, such as Google's Accessibility Scanner [33] and Apple's Accessibility Inspector [22]. These tools operate as static hierarchy analyzers, inspecting declared properties in layout files (*e.g.*, XML [12]) to verify compliance with design guidelines, such as minimum target size. However, since an element's true interactive boundary is a dynamic, runtime construct [13, 27, 52], these static analyzers are inherently incapable of detecting VIMs. This leaves a critical gap in quality assurance (QA) pipeline: developers lack practical, automated tools to diagnose VIMs.

To this end, we propose `TouchAlign`, the first automated, black-box tool for detecting VIMs. Our key insight is that an element's interactive boundary cannot be reliably *inferred* through static analysis, it can only be empirically *observed* through the app's runtime



(a) Shrunken "Add to Cart" Button

(b) Bloated Address Field Area

(c) Offset "Delete" Button

Figure 1: VIM Examples.

The red bounding boxes represent interactive areas, while the green bounding boxes highlight the visual boundaries.

behavior. Specifically, `TouchAlign` operates in a three-stage process. First, it leverages an object detection model to extract the *visual boundaries* of salient elements directly from the screen's rendered pixels. Second, for each candidate element, `TouchAlign` performs a series of systematic, simulated taps in and around its visual boundary. By monitoring the GUI's semantic change, `TouchAlign` maps out the invisible *interactive boundary*. Finally, these two boundaries are compared to identify and report VIMs.

We conducted comprehensive experiments to evaluate the effectiveness of `TouchAlign`. Ablation studies confirm its superior performance and demonstrate the benefits of its dynamic probing strategy. `TouchAlign` accurately identifies interactive boundaries with an average Intersection over Union (IoU) of 0.896, and effectively detects VIMs with a recall of 0.857, significantly outperforming existing baselines. Since its deployment in Meituan's QA pipeline in May 2025, `TouchAlign` has analyzed around 5,000 interactive elements. Of the 104 issues identified by `TouchAlign`, 86 have been confirmed and 85 fixed, thereby substantially improving app usability.

We summarize the contributions of this paper as follows:

- To the best of our knowledge, we are the first to propose and systematically characterize VIM, a pervasive class of GUI defects that, while formerly overlooked, can severely degrade the UX.
- We design and implement `TouchAlign`, a novel black-box tool that combines object detection and dynamic probing to accurately detect VIMs without relying on source code or view hierarchies.
- We evaluate `TouchAlign` on real-world apps and deploy it in Meituan's QA pipeline. Results show that `TouchAlign` effectively uncovers VIMs missed by existing accessibility checkers, thereby improving app usability and UX.

2 BACKGROUND AND MOTIVATION

As apps grow in functionality, their screen layouts become increasingly dense, with numerous interactive elements competing for limited display space. Therefore, balancing visual richness with reliable interaction remains a challenge in mobile GUI design.

In this section, we first introduce common engineering practices employed to manage this trade-off. Then, we discuss their unintended consequences, systematically characterizing the main forms of VIM. Finally, we present an empirical study that highlights the prevalence of these defects in popular, real-world apps.

2.1 Industrial Practice for App Usability

The usability of modern mobile GUIs is largely shaped by *design guidelines*. For instance, Google's Material Design [34] for Android and Apple's Human Interface Guidelines [23] for iOS. These guidelines require minimum touch target sizes and adequate padding to ensure interaction experience. However, in industrial practice, adhering to them poses a significant challenge, especially for feature-rich apps with visually dense GUIs.

To comply with design guidelines in visually dense GUIs, developers often adjust an element's interactive area manually. Such adjustments are typically implemented via Android's `TouchDelegate` mechanism [13] or redirecting touch handling events to invisible containers. According to several industry standards [21, 44], even top-tier mobile developers extensively adopt this straightforward practice.

For implementation, these adjustments are typically associated with specific element types. For example, the interactive area of a small but frequently used "add" icon may be enlarged to improve touch accessibility. Conversely, to prevent accidental touches, adjacent elements often have to shrink their interactive areas to accommodate padding between each other.

2.2 Unintended Spatial Misalignment

These adjustments are bound to decouple an element's visual footprint from its actual interactive region, leading to VIMs under rapid development cycles. Specifically, popular mobile apps are frequently updated, and GUI layouts are routinely refactored as part of agile releases. Since interactive boundaries are invisible and rarely checked, adjustments that initially align with the design intent can gradually fall out of sync. This happens when later changes are made to surrounding layouts or event-handling logic.

Through our study in Section 2.3, we observed that these issues manifest primarily in three forms, each with distinct consequences for UX:

2.2.1 Shrunken Interactive Area. A shrunken interactive area occurs when an element's interactive area is strictly contained within its visual boundary. Although the element appears fully clickable, taps near its edges may be ignored, causing localized interaction failures. As shown in Figure 1(a), a shrunken "Add to Cart" button may ignore edge taps, preventing users from completing expected action.

2.2.2 Bloated Interactive Area. A bloated interactive area extends beyond an element's visual boundary into empty space or adjacent regions. This can trigger unintended actions when users interact with nearby elements or perform scrolling gestures, reducing interaction predictability. Figure 1(b) illustrates an address field whose interactive area spans its parent container, intercepting touches intended for an adjacent "Back" button.

Table 1: Prevalence of VIMs.

App	Category	Downloads	#Screens	#Elements	#VIMs
Taobao 10.56.20	E-commerce	10B+	10	322	7
Pinduoduo 7.91.0	E-commerce	10B+	11	369	4
eBay 6.241.0.1	E-commerce	100M+	7	51	4
Temu 4.24.0	E-commerce	500M+	8	115	5
Wechat 8.0.67	Social	10B+	10	170	2
Weibo 15.12.5	Social	5B+	12	146	5
Rednote 9.17.0	Social	10M+	8	178	4
X 11.56.1-release.0	Social	1B+	6	52	3
Douyin 37.4.0	Video	10B+	9	162	1
Bilibili 8.79.0	Video	5B+	9	189	2
YouTube 21.03.36	Video	10B+	6	104	6
TikTok 43.1.4	Video	1B+	8	114	1
Alipay 10.8.26	Tools	10B+	11	289	2
Dianping 11.56.4	Tools	1B+	10	260	4
Google Translate†	Tools	1B+	4	30	2
ChatGPT 1.2026.013(20)	Tools	500M+	5	44	1
Total	-	-	**134**	**2,361**	**53**

Google Translate Version: 10.1.35.8555527520.1-release.

2.2.3 Offset Interactive Area. An offset interactive area is comparable in size to the visual boundary but spatially misplayed. Taps on the visible element may be ignored, while touches in a nearby region trigger the action, violating users' spatial expectations. Figure 1(c) shows an offset "Delete" button that can be accidentally activated when users attempt to save their shipping address.

While each form of misalignment introduces localized usability risks, their effects can compound in dense GUIs. Bloated or offset areas may overlap adjacent elements, causing ambiguous interactions where a tap unpredictably triggers the wrong target.

2.3 Prevalence Study

To validate that VIM is a broader phenomenon across popular real-world mobile apps rather than an isolated issue specific to Meituan, we conducted an empirical study as a motivating investigation. The goal of this study is to answer the following research question: *Are VIMs prevalent in popular, real-world mobile apps?*

2.3.1 Dataset. To ensure a rigorous and representative assessment, we selected our subjects based on the following criteria. (1) *Prevalence.* We selected widely-used apps from 4 major categories (E-commerce, Social, Video, and Tools) based on their market ranking in both Google Play [1] and major Chinese app markets [2], resulting in four apps per category. This ensures coverage of apps with large and diverse user bases. (2) *Criticality.* For each app, we focused on common foundational screens that are essential to users, such as home feeds, search results, and user profiles. These screens are typically information-dense and serve as standard GUI templates in modern mobile app design. Following these criteria, we collected a total of 16 apps and captured 4–12 screens per app, resulting in a dataset of 134 unique screens and 2,361 interactive elements.

2.3.2 Methodology. The inspection was conducted independently by two of the authors to ensure reliability. For each screen, the authors performed dense interactive probing by systematically executing taps within the visual boundaries of elements and their immediate surroundings. An element was conservatively labeled as a VIM instance if either of the following conditions was satisfied: (1) taps within its visible boundary failed to trigger the intended

[1] https://play.google.com/store
[2] https://app.mi.com

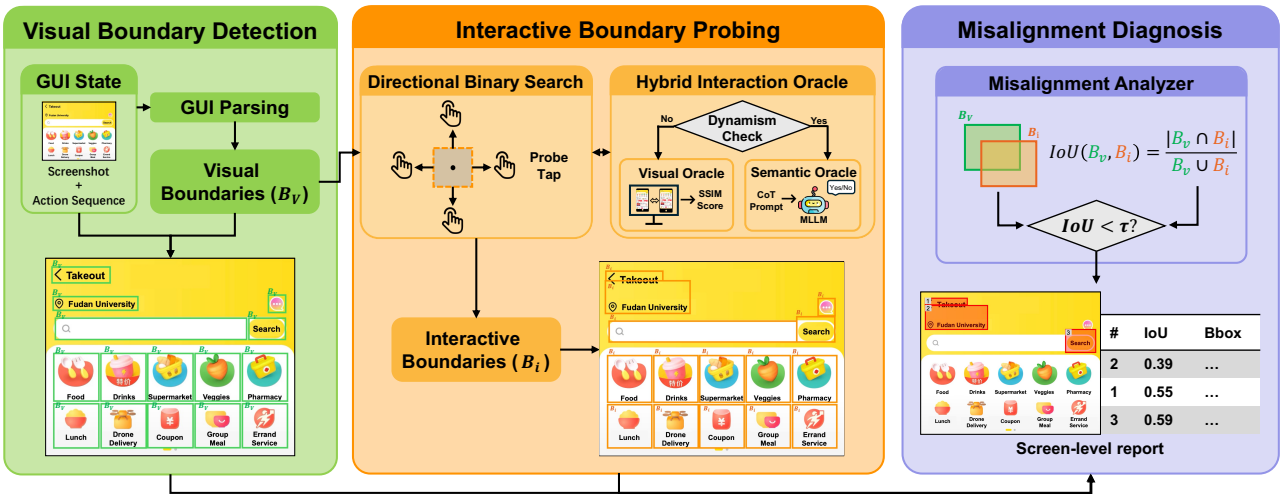


Figure 2: The workflow of TouchAlign.

response, or (2) taps outside its visible boundary triggered the element's associated action. To minimize ambiguity, we focused exclusively on elements with clear visual affordances (*e.g.*, standalone buttons and icons) and excluded composite widgets. The inter-annotator agreement for VIM identification reached a Cohen's Kappa (κ) of 0.87, indicating almost perfect agreement [7, 25]. All remaining discrepancies were resolved through discussion to reach a consensus.

2.3.3 Findings. Our study reveals that VIM is not an isolated defect tied to a specific app or feature, but a recurring phenomenon in modern mobile GUIs. As shown in Table 1, across all 16 studied apps spanning four categories, we identified 53 distinct VIM instances among 2,361 interactive elements.

Furthermore, prior work based on large-scale user reviews provides complementary evidence that issues closely related to VIMs are noticeable in practice. For example, an analysis of 1,447 Google Play reviews of popular apps reports complaints such as "Tapbox overlap to somewhere" and "The buttons are too small to people with sight problem" [6]. Although these reports are not tied to the specific apps studied in our dataset, they offer a user-perspective signal that such issues are difficult to systematically identify during development and testing.

Taken together, this empirical evidence highlights limitations in existing QA workflows and motivates the need for automated techniques that can systematically detect VIMs at scale.

3 TOOL DESIGN

3.1 Overview

Figure 2 illustrates the workflow of TouchAlign for automated detection of VIM in mobile GUIs. The input to TouchAlign is a reproducible GUI state, consisting of (1) a GUI screenshot and (2) the action sequence required to reproduce the state on the Application Under Test (AUT).

Given a target GUI state, TouchAlign operates in three phases. First, in the Visual Boundary Detection phase, TouchAlign analyzes the screenshot using a GUI parsing model to identify interactive GUI elements and extract their visual boundaries. Second, in the Interactive Boundary Probing phase, TouchAlign replays the GUI state on the live AUT and infers each element's interactive boundary through systematic touch probing. Finally, in the Misalignment Diagnosis phase, TouchAlign compares the two boundaries and generates defect reports.

3.2 Visual Boundary Detection

The visual boundary detector extracts the visual boundaries of visually salient GUI elements from a screenshot. These boundaries serve two purposes: (1) defining the target areas for subsequent interactive boundary probing, and (2) providing the visual ground truth for misalignment analysis.

To address the challenges posed by complex GUIs, TouchAlign employs a state-of-the-art GUI parsing model, OmniParser V2 [55]. Given a screenshot, OmniParser detects interactive GUI elements and their bounding box coordinates. The output of this phase is a set of visual boundaries, each corresponding to a distinct interactive element.

3.3 Interactive Boundary Probing

After identifying visual boundaries, TouchAlign determines the corresponding interactive boundaries, which are invisible and only manifest through runtime behavior. To address this, the *Interactive Boundary Prober* integrates two components: (1) *Directional Binary Search*: a probing strategy that incrementally "feels out" the interactive boundary of an element by issuing probe taps and analyzing GUI responses; and (2) *Hybrid Interaction Oracle*: an interaction oracle that evaluates each probe's outcome to determine if it triggered the target element's intended function.

3.3.1 Directional Binary Search. The probing process begins by establishing a reference for the target element's runtime behavior. First, the *Interactive Boundary Prober* records the initial GUI screen as the source screen (S_{source}). Then, it performs a tap at the *anchor point*, which is the geometric center of the element's visual area, and captures the resulting screen. This captured screen is designated

as the reference outcome screen (S_{ref}). Based on the reasonable assumption that production apps released through standard QA pipelines can correctly handle taps at an element's geometric center, S_{ref} can therefore be treated as the ground-truth representation of an on-element interaction. Together, S_{source} and S_{ref} define the reference GUI transition for this element.

Once this behavioral reference is established, the prober searches for the boundary along several predefined directions (*i.e.*, up, down, left, right) starting from the *anchor point*. For each direction, a binary search is performed to locate the farthest position from the anchor that still leads to S_{ref}. The logic for searching one direction is shown in Algorithm 1.

Algorithm 1 Single-Direction Binary Search

Input: $anchor, direction, S_{source}, S_{ref}$
Output: Boundary point $p_{boundary}$
$low \leftarrow anchor, high \leftarrow$ screenEdge
while $high >= low$ **do**
  $p_{probe} \leftarrow$ midpoint$(low, high)$
  $S_{probe} \leftarrow$ tapAndCapture(p_{probe})
  if Oracle$(S_{source}, S_{ref}, S_{probe})$ **then**
    $low \leftarrow p_{probe}$ ▷ On-element, expand outwards
  else
    $high \leftarrow p_{probe}$ ▷ Off-element, shrink inwards
return low

The prober conducts a binary search along a given direction. At each step, a probe tap is issued at the midpoint between the current search ranges, and the resulting screen S_{probe} is evaluated by the *Hybrid Interaction Oracle*. The oracle determines whether the probe corresponds to an on-element interaction. If so, the search expands outward; otherwise, it shrinks inward. This process continues until the boundary point for that direction is located with pixel-level precision. After repeating this for all directions, the collection of discovered boundary points defines the vertices of the inferred interactive boundary.

3.3.2 Hybrid Interaction Oracle. The accuracy of our probing strategy hinges on a reliable oracle, which determines whether a probe tap produced the intended on-element interaction. Given the pre-interaction screen S_{source}, the reference post-interaction screen S_{ref}, and the probe result screen S_{probe}, the oracle outputs a boolean decision: *on-element* or *off-element*. The core criterion for this decision is *functional-semantic equivalence*: the GUI state transition triggered by the probe tap must be functionally and semantically identical to the reference transition.

A straightforward *Visual Oracle* assumes that functionally equivalent screens are visually similar. This oracle computes the Structural Similarity Index (SSIM) [49] between S_{ref} and S_{probe}. A high similarity score suggests an on-element interaction. However, this assumption fails on *dynamically-updating content screens* (*e.g.*, product recommendation feeds), where content changes per visit, rendering pixel-level comparison unreliable.

Task:
You are given screenshots from an interaction probing process on a mobile UI. Your task is to determine whether a probe tap triggers the same functional outcome as a confirmed on-element interaction.

Input:
Three screenshots are provided in sequence:
- S_{source}: the screen before any interaction.
- S_{ref}: the screen after a confirmed on-element tap on the target element.
- S_{probe}: the screen after a probe tap at a candidate location.

Decision Procedure (Two-Stage Reasoning):
1. **Infer Target Function** Compare S_{source} and S_{ref} to identify the primary functional effect of the interaction (e.g., navigation, selection, state change). Based on this transition, infer the core function of the target element.
2. **Evaluate Probe Consistency** Compare S_{source} and S_{probe} to determine whether the observed transition is functionally and semantically equivalent to the inferred target function.
- Decide "Yes" if the core functional effect is identical.
- Decide "No" if S_{probe} is visually and functionally identical to S_{source}, or if it triggers a different interaction.

Handling Dynamic Content:
During the Decision Procedure, you must ignore visual differences caused by dynamically updating elements, including recommendation feeds, rotating banners, advertisements, and decorative UI components.

Output:
Return a JSON object with the following fields:

```
{
    Decision: "Yes" or "No",
    Reasoning: "Step-by-step explanation following
                the decision procedure above."
}
```

Figure 3: The Chain-of-Thought prompt for the Semantic Oracle.

To address this, TouchAlign incorporates a *Semantic Oracle* leveraging Multimodal Large Language Models (MLLMs) like GPT-4o [37]. The oracle operates through a carefully constructed chain-of-thought prompt [50] shown in Figure 3. Given the three screenshots, the prompt first directs the model to analyze the reference transition (S_{source} to S_{ref}) and infer the target element's function. It then instructs the model to evaluate if the probe transition (S_{source} to S_{probe}) is consistent with that inferred function. This approach enables the model to focus on functional semantics, effectively ignoring irrelevant visual differences caused by dynamic content. The final "Yes/No"decision made by the MLLM serves as the oracle's output.

To combine the high accuracy of the Semantic Oracle with the efficiency of the Visual Oracle, TouchAlign implements a *Hybrid*

Oracle Strategy. Before probing begins, a pre-processing step assesses the dynamism of the target screen. It triggers an on-element tap at the anchor twice to generate two outcome screens, then computes their visual similarity using SSIM. A high SSIM score indicates a "static" screen, for which the Visual Oracle is employed. Conversely, a low SSIM score indicates a "dynamic" screen, prompting the use of the more robust Semantic Oracle. This adaptive strategy allows TouchAlign to balance accuracy and efficiency in large-scale industrial testing.

3.4 Misalignment Diagnosis

The final phase of the TouchAlign workflow diagnoses misalignment based on the detected boundaries and generates defect reports. This process is handled by the *Misalignment Analyzer*, which takes an element's visual boundary B_v and its inferred interactive boundary B_i as input.

The core of our detection logic involves quantifying the spatial discrepancy between these two boundaries using the Intersection over Union (IoU) metric. For a given element, the misalignment score is calculated as:

$$IoU(B_v, B_i) = \frac{|B_v \cap B_i|}{|B_v \cup B_i|} \tag{1}$$

A misalignment defect is flagged if the IoU falls below a predefined threshold τ (*i.e.*, $IoU < \tau$), indicating that the visual and interactive regions are sufficiently disjoint to potentially degrade UX. The output is a screen-level report that highlights all misaligned elements on each screen, by overlaying the inferred interactive boundaries on the original screenshot. Since the IoU score directly reflects the severity of VIM, reported defects are ranked in ascending order of IoU, allowing developers to prioritize the most severe issues.

4 EVALUATION

In this section, we evaluate the performance of TouchAlign by answering the following research questions (RQs):

- **RQ1**: How accurate can the Interactive Boundary Probing module of TouchAlign detect interactive boundaries?
- **RQ2**: How effective is TouchAlign in detecting VIMs?
- **RQ3**: To what extent can TouchAlign detect VIMs in real-world mobile apps?

4.1 Evaluation Setup

4.1.1 Dataset. To the best of our knowledge, no publicly available benchmark exists for evaluating VIM. Therefore, we construct a benchmark dataset ourselves based on the Meituan app. The app's structure, which integrates dozens of distinct business lines (*e.g.*, Takeout, Travel) developed by different teams, provides a naturally diverse and representative dataset of GUI design styles.

We evaluate TouchAlign across five business lines (*i.e.* Takeout, Dining, Travel, Grocery and Video), each selected to represent specific GUI challenges and ensure data diversity. To comprehensively cover real-world scenarios, we selected the entry pages from each business line and ran the STOAT tool [41] for 30 minutes to collect GUI states. The collected states were manually reviewed by two authors to remove duplicates and irrelevant screens. This process resulted in 65 GUIs and 1,745 interactive elements, with an average of 26.85 elements per screen (ranging from 9 to 51). Table 2 summarizes the dataset, covering a wide range of real-world scenarios.

Table 2: Details of the Dataset.

Business Line	#GUIs	#Ele.	Avg. Ele.	Max. Ele.	Min. Ele.
Takeout	13	445	34.23	51	15
Dining	16	392	24.50	37	14
Travel	17	427	25.12	45	9
Grocery	14	394	28.14	38	13
Video	5	87	17.40	22	12
Total	65	1,745	26.85	51	9

For **RQ1**, four authors independently annotated the interactive boundaries for all 1,745 elements to collect ground-truth, resulting in 6,980 individual annotations. This tedious process required substantial manual effort and lasted for over two weeks. We first evaluated the alignment of the annotations by calculating the average IoU, which reached a high value of 0.932. To ensure high confidence in our ground truth, we conservatively defined the final interactive boundary for each element as the intersection of all the independent annotations.

For **RQ2**, to evaluate the effectiveness of TouchAlign, we manually injected VIMs into 175 elements ($\approx$ 10% of the dataset). To ensure the injected VIMs reflect real-world scenarios, the injection was guided by historical user complaints about VIMs in Meituan. For each screen, 1 to 10 elements were randomly selected for injection. We modified either the visual boundaries or the interactive boundaries of selected elements:

- Shrunken / Bloated VIM: We kept the element's center point fixed and either shrank (or expanded) its interactive boundary, or enlarged (reduced) its visual boundary.
- Offset VIM: We either bound the element's interactive event to a parent container with a different center, or adjusted its visual size and position so that the element's visual center no longer aligned with its interactive center.

By systematically applying these modifications, our injected VIMs capture the three common real-world misalignment patterns, allowing us to rigorously test TouchAlign's detection capabilities.

4.1.2 Baselines. To highlight our contribution, in **RQ1**, we evaluate TouchAlign against three representative baselines. These baselines are designed to assess the necessity of our core modules through systematic ablation:

- Google Accessibility Scanner [33]: a widely-adopted industrial tool for diagnosing accessibility issues. As it relies solely on the static Accessibility Tree to determine interactive areas, it can be viewed as an ablated version of TouchAlign that omits the entire *Directional Binary Search* strategy.
- TouchAlign (w/o Semantic Oracle): This variant replaces our Hybrid Oracle with the Visual Oracle. This ablation isolates the impact of semantic reasoning in handling non-deterministic, dynamic GUI screens.
- TouchAlign (w/o Visual Oracle): This variant exclusively employs the MLLM-based Semantic Oracle for all decisions.

Table 3: Performance and Efficiency Comparison of TouchAlign against Baselines.

Method	Takeout			Dining			Travel			Grocery			Video			Avg.		
	IoU ↑	Time ↓	Cost ↓	IoU ↑	Time ↓	Cost ↓	IoU ↑	Time ↓	Cost ↓	IoU ↑	Time ↓	Cost ↓	IoU ↑	Time ↓	Cost ↓	IoU ↑	Time ↓	Cost ↓
Google Accessibility Scanner	0.737	**0.126**	**0.000**	**0.989**	**0.147**	**0.000**	0.754	**0.185**	**0.000**	0.209	**0.171**	**0.000**	**0.840**	**0.259**	**0.000**	0.706	**0.162**	**0.000**
TouchAlign (w/o Semantic Oracle)	0.616	52.811	**0.000**	0.720	50.136	**0.000**	0.582	50.793	**0.000**	0.873	51.622	**0.000**	0.374	54.289	**0.000**	0.633	51.521	**0.000**
TouchAlign (w/o Visual Oracle)	**0.959**	314.476	0.018	0.905	322.294	0.018	0.881	325.142	0.018	0.939	315.983	0.018	0.764	319.172	0.018	**0.914**	319.417	0.018
TouchAlign (Hybrid Oracle)	0.894	128.694	0.004	0.910	107.289	0.005	**0.899**	149.861	0.004	**0.951**	112.425	0.008	0.825	266.195	0.014	0.896	132.247	0.005

It serves to evaluate the efficiency-accuracy tradeoff and the necessity of the adaptive hybrid strategy.

4.1.3 Metrics. For **RQ1**, we employ three complementary metrics to access the accuracy and efficiency of the *Interactive Boundary Probing* module. First, we quantify the accuracy using IoU between the detected interactive boundary B_d and the ground-truth interactive boundary B_i. A higher IoU value directly indicates greater detection accuracy. Then, we quantify the module's efficiency using the average latency (s) and monetary cost ($) per element. The latency accounts for the total time consumed by the *Directional Binary Search* process, excluding screen load times. The monetary cost is calculated based on the MLLM tokens consumed during the probing process, following the official pricing of Qwen [3].

For **RQ2**, we formulate VIM detection as a binary classification problem: an element is classified as misaligned (Positive) if the IoU between its detected visual and interactive boundaries falls below a threshold τ. This threshold is based on our analysis of over three months of historical VIM-related user complaints at Meituan. Given that the overlap between visual and interactive areas (measured by IoU) approximately follows a normal distribution, we applied a standard P_{90} quantile [1] to define the misalignment threshold. Specifically, historical data shows that over 90% of user-reported interaction failures corresponded to VIMs with an IoU below 0.65. Accordingly, we set the misalignment threshold $\tau = 0.65$ to ensure our analysis focuses on defects severe enough to cause user-perceptible issues. We report Precision, Recall, F1-score, and False Positive Rate to evaluate detection effectiveness.

4.1.4 Implementation. TouchAlign works by obtaining GUI screenshots via UIAutomator2 [3] and performing simulated interactions through the Android Debug Bridge (ADB) [2]. And we set the SSIM threshold θ within the Hybrid Oracle module to 0.8 to balance MLLM use. For the Semantic Oracle, we adopt Qwen3-VL-Flash [4], which offers superior performance while maintaining fast response speeds. All experiments were executed on a MacBook Pro equipped with a 2.6 GHz 6-core Intel Core i7 processor. The AUT (Meituan V12.50.203) was run on two Android devices(*e.g.* Xiaomi 12, Samsung Galaxy A53) configured to a standard 1080x2400 resolution to ensure a consistent analysis environment.

4.2 RQ1: Interactive Area Accuracy

To answer RQ1, we compare the interactive boundaries detected by TouchAlign against the ground truth annotations within our benchmark and baselines. Table 3 presents the result across the five business lines.

[3]https://modelstudio.console.alibabacloud.com/ap-southeast-1/?tab=doc#/doc/?type=model&url=2840914_2&modelId=qwen3-vl-flash

Table 4: VIM detection performance of TouchAlign.

	Takeout	Dining	Travel	Grocery	Video	Aggregate
#Elements	445	392	427	394	87	1,745
#VIMs	36	41	47	36	15	175
#TP	32	33	41	33	11	150
#FP	9	6	8	4	6	33
#TN	400	345	372	354	66	1537
#FN	4	8	6	3	4	25
Precision	0.780	0.846	0.837	0.892	0.647	0.820
Recall	0.889	0.805	0.872	0.917	0.733	0.857
F_1	0.831	0.825	0.854	0.904	0.687	0.838
FPR	0.022	0.017	0.021	0.011	0.083	0.021

TouchAlign achieves the best of both worlds: achieving an average IoU of 0.896 with moderate time and monetary overhead. In terms of detection accuracy, both Google Accessibility Scanner and the pure Visual Oracle variant show clear weaknesses. Google Accessibility Scanner performs particularly poorly on the Grocery business line (IoU = 0.209). This limitation stems from its reliance on accessibility trees, whose quality heavily depends on implementation practices. As many Grocery pages are built using with the HTML5 technique [16], which leads to incomplete or noisy view hierarchies. As a result, the tool struggles on business lines with inconsistent structural metadata (*e.g.*, Takeout, Travel), while performing well only when the view hierarchy is clean and well-structured (*e.g.*, Video).

The pure Visual Oracle variant also exhibits unstable performance, especially on dynamic screens (IoU = 0.374 on Video pages). Even on partially dynamic, content-focused pages (*e.g.*, product listing pages in Takeout), its performance degrades, as recommendation algorithms cause products to change between visits, undermining SSIM-based detection. This result demonstrates that visual similarity alone is insufficient for reliable boundary inference in real-world apps and highlights the necessity of semantic reasoning.

Compared with the pure Semantic Oracle variant, TouchAlign maintains comparable accuracy (IoU 0.896 vs. 0.914) while significantly reducing overhead, requiring 58% less execution time and 72% lower monetary cost on average. This confirms that the hybrid strategy successfully balances the high accuracy of MLLM-based reasoning with the efficiency of visual analysis, making TouchAlign practical for large-scale industrial deployment. Besides, although Google Accessibility Scanner exhibits negligible runtime and zero monetary cost, its average IoU (0.706) is substantially lower and highly unstable across business lines.

4.3 RQ2: Misalignment Detection Effectiveness

As shown in Table 4, TouchAlign achieves a precision of 0.820, a recall of 0.857, and a false positive rate (FPR) of 0.021 under the IoU threshold $\tau = 0.65$.

Table 5: Performance and Impact of TouchAlign in Real-world Industrial Deployment.

Business Line	#GUIs	#Elements	#VIMs	#Confirmed VIMs	#Fixed VIMs
Takeout	65	1,040	27	23	22
Grocery	72	1,116	18	15	15
Platform	30	1,115	10	10	10
Hotel	48	1,449	30	24	24
Medicine	12	258	19	14	14
Total	227	4,978	104	86	85

Platform refers to common pages (*e.g.*, home screens, user profiles) that are shared across the super-app rather than belonging to a specific business line.

The high recall indicates that TouchAlign successfully identifies the vast majority of misaligned elements. This property is particularly important in large-scale industrial settings, where undetected VIMs may directly translate into user-sensible interaction failures and subsequent complaints. By maintaining comprehensive coverage, TouchAlign serves as a reliable early-warning mechanism for potential usability issues.

At the same time, the high precision and low FPR show that most elements flagged as misaligned are indeed true defects. Although false positives still exist, their practical impact is limited. In deployment, TouchAlign visualizes the interactive boundaries of GUI elements by overlaying colored bounding boxes in defect reports, making it easy for developers to distinguish true from false positives at a glance.

Importantly, we intentionally keep the threshold τ fixed rather than tuning it to further reduce false positives. This reflects a practical trade-off: false negatives are more costly than false positives. Missing a VIM risks real user complaints, whereas false alarms can be efficiently filtered through lightweight human inspection.

Finally, the results remain consistent across different business lines, demonstrating TouchAlign's generalizability.

4.4 RQ3: Industrial Deployment

In large-scale super-apps such as Meituan, multiple business lines are typically developed and maintained by decentralized teams. As described in Section 1, to comply with design guidelines regarding minimum touch targets and inter-element spacing, these teams often manually calibrate the interactive boundaries of GUI elements. However, due to variations in development practices across teams, such calibration often lacks consistency. As a result, visually identical elements (*e.g.*, navigation icons) may exhibit substantially different interactive areas across business lines. Such inconsistency degrades the UX, as it imposes an unnecessary cognitive load by forcing users to adapt to the interactive behaviors of each specific business line.

Therefore, ever since TouchAlign's release in May 2025, it has been integrated into Meituan's automated testing platform and made directly accessible to QA engineers. Every time a new version is built, QAs launch the Meituan app on remote real devices provided by the testing platform and invoke TouchAlign on selected pages. For each page, TouchAlign automatically analyzes all interactive GUI elements by comparing their visual boundaries with interactive boundaries inferred through systematic tap simulation. Elements whose IoU falls below the predefined threshold $\tau = 0.65$ are reported as potential VIMs and submitted to developers for evaluation and confirmation.

Between May 2025 and December 2025, TouchAlign was applied across multiple major business lines, focusing on high-impact entry pages exceeding 100k daily visits. In total, TouchAlign analyzed 227 GUIs containing 4,978 interactive elements, and reported 104 potential VIMs. Among them, 86 issues (82.7%) were confirmed by developers as genuine defects, and 85 confirmed issues have already been fixed. Table 5 summarizes the detailed results across business lines. Notably, TouchAlign demonstrates consistently high confirmation and fix rates across different business lines, indicating stable performance under heterogeneous development settings. Moreover, we further analyze the root causes for all the confirmed VIMs. In practice, bloated interactive areas are most common (67 issues), typically caused by binding interactive areas to parent containers (*e.g.*, Android ViewGroup), while shrunken (17 issues) and offset Interactive Areas (2 issues) arise from various implementation oversights.

From the QA perspective, TouchAlign substantially improves inspection efficiency. With TouchAlign, a QA engineer can review approximately 160 reported elements per person-day, each of which has a high likelihood of being a true defect, rather than exhaustively probing all elements. In contrast, without TouchAlign, manual exploratory testing typically inspects around 72 elements per person-day, most of which do not exhibit any misalignment issues. As a result, TouchAlign significantly improves the efficiency of identifying actionable VIMs by prioritizing high-risk elements for manual review. Developers report that confirmed issues can be fixed without violating existing design guidelines on touch target size or spacing. Issues that remain unconfirmed often require additional discussion with UI design teams, as fixing them may conflict with existing design guidelines. Overall, these results suggest that TouchAlign not only scales to real-world industrial workloads but also delivers actionable and cost-effective feedback within existing quality assurance pipelines.

5 CASE STUDY AND LESSONS LEARNED

5.1 Case Study: VIM in Navigation Bars

During the 8 months of TouchAlign's deployment, we observed a notable pattern: a large proportion (61.5%) of detected VIMs were concentrated in navigation bars. As shown in Figure 4, navigation bars are central to user navigation and contain dense collections of small interactive icons, such as back, search, and share.

(a) Overlap Between Offset and Bloated Elements. (b) Overlap Between Shrunken and Bloated Elements.

Figure 4: Misalignment in Navigation Bars.

Yet, in practice, small icons often receive less testing attention compared to primary call-to-action elements (*e.g.*, "Buy Now", "Checkout") [28]. This imbalance creates a vulnerability: navigation bars hold long-term value for user retention, but their reliability is undermined by insufficient testing attention.

Figure 4 illustrates two representative defects. In one case, the bloated interactive boundary of the "Share" icon exacerbated the interaction difficulty for the adjacent "Favorite" icon, whose interactive area was offset. In another, an expanded hit-area for "More" button overlapped the neighboring search button, leading to frequent mis-taps.

Such defects are insidious: since the core functionality generally remains accessible via some portion of the element, they can easily go undetected even by experienced testers, while subtly undermining the UX. This elusive nature explains why the issues uncovered by `TouchAlign` were immediately acknowledged and promptly fixed by developers.

This case study highlights that navigation elements deserve testing priority comparable to transactional controls. Although defects in navigation bars may not block core tasks, they accumulate friction in routine interactions and erode long-term usability.

Our analysis leads to two actionable recommendations for engineering teams: (1) prioritizing navigation bars and other high-density control regions for automated spatial alignment testing, and (2) using well-defined, reusable components with explicit interactive boundary specifications to support systematic validation. These measures would shift defect detection from ad-hoc discovery to systematic prevention, strengthening the long-term usability of mobile applications.

5.2 Lessons Learned: Structural Debt in GUI Development

Our investigation into VIM reveals a deeper systemic issue in mobile app development: the unreliability of structured GUI metadata [27]. In principle, misalignment detection could rely on comparing an element's visual appearance with its declared properties in structural metadata (*e.g.*, the view hierarchy). In practice, however, such metadata is often noisy or incomplete [11, 26], making this approach unreliable.

This unreliability largely stems from common engineering practices driven by rapid development. We identified three primary causes: (1) *Hybrid Frameworks*, where WebViews and native components are mixed, breaking the propagation of structural information [39]; (2) *View Overlays*, where new views are layered over existing elements for implementation simplicity, obscuring them in the hierarchy [11, 26]; and (3) *Performance Optimizations*, where multiple logical controls are flattened into a single custom-drawn view, collapsing their distinct nodes in the hierarchy. As illustrated in Figure 5, these practices, while offering short-term benefits, result in a form of *structural debt*: the foundational metadata no longer faithfully represents the user interface.

(a) Noisy View Hierarchy (b) Incomplete View Hierarchy

Figure 5: Examples of Unreliable GUI Metadata.

The consequences of structural debt are twofold. First, it directly harms accessibility, as users relying on assistive technologies may find elements un-focusable or entirely missing. Second, for a wide range of GUI modeling tasks, this noisy metadata has necessitated a body of research focused on denoising and repairing GUI layouts [11, 26]. These efforts, while valuable, are fundamentally compensatory; they attempt to repair flawed metadata after the fact rather than prevent its creation.

The design of `TouchAlign` is therefore a response to this imperfect reality. By operating directly on pixels and runtime behavior, it bypasses unreliable structural metadata and avoids the debt-and-repair cycle. Our findings suggest that greater attention must be paid to the integrity of structured GUI metadata. A "clean" accessibility tree should not be treated as a feature for *accessibility*, but as a foundational component for ensuring an application's broader *usability and testability*. While behavior-driven tools like `TouchAlign` provide a robust fallback, the most sustainable path forward lies in addressing the root cause: eliminating structural debt at its source.

5.3 Threats to Validity

Internal Validity. One potential threat arises from `TouchAlign`'s reliance on third-party machine learning components, including the visual boundary detector (OmniParser) and the MLLM-based oracle. Their performance directly affects `TouchAlign`'s accuracy. To mitigate this, we selected a state-of-the-art GUI parsing model and designed a hybrid oracle that invokes MLLM only for challenging dynamic GUIs. Although our evaluation demonstrates high practical accuracy, advances in these underlying models may further improve `TouchAlign`'s performance.

External Validity. Another potential concern is whether VIM is specific to Meituan's development practices. We argue that VIM is an inherent challenge in modern GUI engineering rather than a company-specific defect. As observed in Section 2.3, VIM instances were identified across a wide range of popular apps. These findings suggest that VIM is a widespread yet neglected phenomenon. Its invisible nature allows it to easily bypass conventional automated testing pipelines relying on static code analysis or accessibility trees.

Furthermore, while our evaluation focused on the Meituan app running on Android devices with a unified configuration (1080×2400 resolution), `TouchAlign` is designed as a strictly black-box solution that operates solely on pixel and runtime feedback. It does not rely on proprietary internal mechanisms or platform-specific coding standards, making both the VIM problem and our approach broadly applicable. We leave the investigation of VIM characteristics across heterogeneous device configurations (*e.g.*, varying resolutions) and platforms (*e.g.*, iOS) to future work.

6 RELATED WORK

6.1 GUI Design Violation Detection

Detecting violations of design guidelines in GUIs has been a long-standing research topic. Most existing studies focus on visual design principles [17, 35, 53, 54, 57, 58]. For instance, Seenomaly [58] examines GUI animations (*e.g.*, card movement, menu slide in/out) to ensure compliance with guidelines such as content appearance, elevation, and shadow effects. Minuku [17] addresses display issues like widget occlusion and missing elements, fine-tuning a VLM to detect potential display problems. GVT [35] identifies inconsistencies between design mock-ups and actual implementations, proposing a widget matching algorithm. However, these works are limited to visual design, neglecting interactivity.

On the other hand, some studies focus on the interactive layer of GUIs [15, 29, 33]. Google Accessibility Scanner [33], for instance, ensures that interactive areas are sufficiently large to avoid user confusion. EP-detector [15] identifies error-prone operation anomalies (EPAs), where poorly designed interactive elements lead to unexpected user outcomes. GUIPilot [29] detects process inconsistencies by translating natural language interactions from design mock-ups into executable actions on mobile screens. However, these approaches either overlook the visual side or assume that visual and interactive boundaries are always aligned.

In contrast, `TouchAlign` is the first to address both the visual and interactive layers of GUIs. Our goal is to detect misalignment between the visual boundary and the interactive boundary, bridging the gap between what users see and how they interact with the interface.

6.2 GUI Screen Parsing

LLMs are increasingly adopted for automating GUI tasks [9, 10, 18, 24, 38, 43, 48, 51, 56]. To help LLMs effectively understand GUIs, screen parsing techniques are crucial for detecting and annotating GUI elements.

Most existing approaches rely on parsing the view hierarchy to obtain the bounding boxes of GUI elements. These methods typically filter XML nodes corresponding to interactive elements based on attributes such as `clickable` and `scrollable`, and retrieve bounding boxes from the `bounds` attribute. However, as discussed in Section 5, view hierarchies are often noisy and incomplete. To address this, some works, like MUD [11], utilize deep learning methods alongside human annotators to improve the quality of view hierarchies. ReBL [46] takes a heuristic approach, grouping XML nodes that are likely to correspond to the same GUI element.

Despite these improvements, screen parsing based on view hierarchies still lacks generalizability. As a result, pure vision-based methods have emerged as a promising alternative. Mobile-Agent [47] was the first to leverage visual perception tools to accurately identify and locate both visual and textual elements within GUIs. Similarly, Omniparser [55] fine-tunes specialized models to enhance LLMs' ability to generate actions grounded in specific regions of the interface.

However, these approaches assume that visual and interactive boundaries are always aligned, which is not always the case. Our work highlights this misalignment issue and proposes a method to detect it.

6.3 LLM-based GUI Testing

Beyond automating GUI tasks, LLMs are applied in various aspects of GUI testing. For instance, several studies have used LLMs for bug reproduction, aiming to automatically identify and reproduce bugs based on bug reports [8, 20, 42, 46]. LLMs have also been applied to explore GUIs, promoting better test coverage and identifying bug-prone paths [5, 31, 45]. Additionally, LLMs are used to generate text input, helping automate testing of forms and user inputs [30, 32].

In `TouchAlign`, we leverage the image understanding capabilities of MLLMs to implement the Semantic Oracle sub-module. This oracle calculates the semantic similarity between GUI snapshots, going beyond pixel-level visual matching to reason about the functional semantics of the interface. This innovation is crucial for detecting misalignment, particularly in dynamic or content-heavy applications where visual comparison alone fails to capture the nuances of GUI.

7 CONCLUSION

In this paper, we introduced and systematically characterized visual-interactive misalignment, a pervasive yet overlooked class of defects that violates the spatial predictability of mobile GUIs. To address this critical gap, we presented `TouchAlign`, a black-box approach that empirically detects these defects by quantitatively comparing an element's visual boundary, identified via object detection, with its interactive boundary, mapped out through dynamic probing guided by a hybrid, LLM-powered oracle. Our evaluation and production deployment at Meituan confirm the effectiveness of `TouchAlign`: it detected misalignments with 0.86 recall and uncovered 86 confirmed, critical defects across over 200 real-world screens. Furthermore, our case studies highlight the insidious nature of these defects and demonstrate the practical benefits of using `TouchAlign` to diagnose spatial misalignment in mobile apps.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Project No. 62572127) and Meituan. Y.Zhou is the corresponding author.

References

[1] 2025. Percentile. https://en.wikipedia.org/wiki/Percentile. Accessed: 2025.

[2] Accessed: 2025. *Android Debug Bridge (adb).* https://developer.android.com/tools/adb

[3] Accessed: 2025. *Write automated tests with UI Automator.* https://developer.android.com/training/testing/other-components/ui-automator

[4] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. 2025. Qwen3-VL Technical Report. *arXiv preprint arXiv:2511.21631* (2025).

[5] Mengzhuo Chen, Zhe Liu, Chunyang Chen, Junjie Wang, Boyu Wu, Jun Hu, and Qing Wang. 2025. Standing on the Shoulders of Giants: Bug-Aware Automated GUI Testing via Retrieval Augmentation. *Proc. ACM Softw. Eng.* 2, FSE (2025), 825–846.

[6] Qiuyuan Chen, Chunyang Chen, Safwat Hassan, Zhengchang Xing, Xin Xia, and Ahmed E. Hassan. 2021. How Should I Improve the UI of My App?: A Study of User Reviews of Popular Apps in the Google Play. *ACM Trans. Softw. Eng. Methodol.* 30, 3 (2021), 37:1–37:38.

[7] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20 (1960), 37 – 46.

[8] Sidong Feng and Chunyang Chen. 2024. Prompting Is All You Need: Automated Android Bug Replay with Large Language Models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 67:1–67:13.

[9] Sidong Feng, Changhao Du, Huaxiao Liu, Qingnan Wang, Zhengwei Lv, Gang Huo, Xu Yang, and Chunyang Chen. 2025. Agent for User: Testing Multi-User Interactive Features in TikTok. In *47th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, SEIP@ICSE 2025, Ottawa, ON, Canada, April 27 - May 3, 2025*. IEEE, 57–68.

[10] Sidong Feng, Haochuan Lu, Jianqin Jiang, Ting Xiong, Likun Huang, Yinglin Liang, Xiaoqin Li, Yuetang Deng, and Aldeida Aleti. 2024. Enabling Cost-Effective UI Automation Testing with Retrieval-Based LLMs: A Case Study in WeChat. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*, Vladimir Filkov, Baishakhi Ray, and Minghui Zhou (Eds.). ACM, 1973–1978.

[11] 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 *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*, Florian 'Floyd' Mueller, Penny Kyburz, Julie R. Williamson, Corina Sas, Max L. Wilson, Phoebe O. Toups Dugas, and Irina Shklovski (Eds.). ACM, 7:1–7:14.

[12] Google. 2025. Layouts in Views - Android Developers. https://developer.android.com/develop/ui/views/layout/declaring-layout. Accessed: 2026.

[13] Google. 2025. Manage Touch Events in a ViewGroup - Android Developers. https://developer.android.com/develop/ui/views/touch-and-input/gestures/viewgroup?hl=en. Accessed: 2025.

[14] Google. 2026. Touch target size - Android Accessibility Help. https://support.google.com/accessibility/android/answer/7101858?hl=en. [Accessed 2026].

[15] Chenkai Guo, Qianlu Wang, Naipeng Dong, Lingling Fan, Tianhong Wang, Weijie Zhang, Enbao Chen, Zheli Liu, and Lu Yu. 2025. EP-Detector: Automatic Detection of Error-Prone Operation Anomalies in Android Applications. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2739–2750.

[16] Ian Hickson. 2011. *HTML5: A Vocabulary and Associated APIs for HTML and XHTML*. Working Draft WD-html5-20110525. World Wide Web Consortium (W3C). https://www.w3.org/TR/2011/WD-html5-20110525/ Accessed: 2025.

[17] Yongxiang Hu, Ke Liu, Hailiang Jin, Shiyu Guo, Juxing Yuan, Xin Wang, and Yangfan Zhou. 2025. Minuku: Detecting Diverse Display Issues in Mobile Apps with Small-scale Dataset. In *ASE 2025 Industry Showcase, 40th IEEE/ACM International Conference on Automated Software Engineering (ASE 2025), Seoul, South Korea, November 16-20, 2025*. To appear.

[18] Yongxiang Hu, Xuan Wang, Yingchuan Wang, Yu Zhang, Shiyu Guo, Chaoyi Chen, Xin Wang, and Yangfan Zhou. 2024. AUITestAgent: Automatic Requirements Oriented GUI Function Testing. *CoRR* abs/2407.09018 (2024).

[19] Yongxiang Hu, Yu Zhang, Xuan Wang, Yingjie Liu, Shiyu Guo, Chaoyi Chen, Xin Wang, and Yangfan Zhou. 2025. KuiTest: Leveraging Knowledge in the Wild as GUI Testing Oracle for Mobile Apps. In *47th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, SEIP@ICSE 2025, Ottawa, ON, Canada, April 27 - May 3, 2025*. IEEE, 34–45.

[20] Yuchao Huang, Junjie Wang, Zhe Liu, Mingyang Li, Song Wang, Chunyang Chen, Yuanzhe Hu, and Qing Wang. 2025. One Sentence Can Kill the Bug: Auto-Replay Mobile App Crashes From One-Sentence Overviews. *IEEE Trans. Software Eng.* 51, 4 (2025), 975–989.

[21] Huawei. 2025. Application UX Standards for General Devices. https://developer.huawei.com/consumer/en/doc/design-guides/ux-guidelines-general-0000001760708152#section1950204881216. Accessed: 2026.

[22] Apple Inc. 2024. Accessibility Inspector - Apple Developer Documentation. https://developer.apple.com/documentation/accessibility/accessibility-inspector. Accessed: 2025.

[23] Apple Inc. 2024. Human Interface Guidelines - Apple Developer. https://developer.apple.com/design/human-interface-guidelines. Accessed: 2025.

[24] Qichao Kong, Zhengwei Lv, Yiheng Xiong, Dingchun Wang, Jingling Sun, Ting Su, Letao Li, Xu Yang, and Gang Huo. 2025. ProphetAgent: Automatically Synthesizing GUI Tests from Test Cases in Natural Language for Mobile Apps. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, FSE Companion 2025, Clarion Hotel Trondheim, Trondheim, Norway, June 23-28, 2025*, Leonardo Montecchi, Jingyue Li, Denys Poshyvanyk, and Dongmei Zhang (Eds.). ACM, 174–179.

[25] J Richard Landis and Gary G. Koch. 1977. The measurement of observer agreement for categorical data. *Biometrics* 33 1 (1977), 159–74.

[26] Gang Li, Gilles Baechler, Manuel Tragut, and Yang Li. 2022. Learning to Denoise Raw Mobile UI Layouts for Improving Datasets at Scale. In *CHI '22: CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, 29 April 2022 - 5 May 2022*, Simone D. J. Barbosa, Cliff Lampe, Caroline Appert, David A. Shamma, Steven Mark Drucker, Julie R. Williamson, and Koji Yatani (Eds.). ACM, 67:1–67:13.

[27] Gang Li and Yang Li. 2023. Spotlight: Mobile UI Understanding using Vision-Language Models with a Focus. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

[28] Linlin Li, Ruifeng Wang, Xian Zhan, Ying Wang, Cuiyun Gao, Sinan Wang, and Yepang Liu. 2023. What You See Is What You Get? It Is Not the Case! Detecting Misleading Icons for Mobile Applications. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, René Just and Gordon Fraser (Eds.). ACM, 538–550.

[29] Ruofan Liu, Xiwen Teoh, Yun Lin, Guanjie Chen, Ruofei Ren, Denys Poshyvanyk, and Jin Song Dong. 2025. GUIPilot: A Consistency-Based Mobile GUI Testing Approach for Detecting Application-Specific Bugs. *Proc. ACM Softw. Eng.* 2, ISSTA (2025), 753–776.

[30] Zhe Liu, Chunyang Chen, Junjie Wang, Xing Che, Yuekai Huang, Jun Hu, and Qing Wang. 2023. Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 1355–1367.

[31] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2024. Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 100:1–100:13.

[32] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Yuekai Huang, Jun Hu, and Qing Wang. 2024. Unblind Text Inputs: Predicting Hint-text of Text Input in Mobile Apps via LLM. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*, Florian 'Floyd' Mueller, Penny Kyburz, Julie R. Williamson, Corina Sas, Max L. Wilson, Phoebe O. Toups Dugas, and Irina Shklovski (Eds.). ACM, 51:1–51:20.

[33] Google LLC. 2024. Android Accessibility Features - Google Support. https://support.google.com/accessibility/android/answer/6376570. Accessed: 2025.

[34] Google LLC. 2024. Designing: Structure - Material Design 3. https://m3.material.io/foundations/designing/structure. Accessed: 2025.

[35] Kevin Moran, Boyang Li, Carlos Bernal-Cárdenas, Dan Jelf, and Denys Poshyvanyk. 2018. Automated reporting of GUI design violations for mobile apps. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 165–175.

[36] Jakob Nielsen. 1994. Enhancing the explanatory power of usability heuristics. In *Conference on Human Factors in Computing Systems, CHI 1994, Boston, Massachusetts, USA, April 24-28, 1994, Conference Companion*, Catherine Plaisant (Ed.). ACM, 210.

[37] OpenAI. 2024. GPT-4o. https://openai.com/index/hello-gpt-4o/. Accessed: 2025.

[38] Dezhi Ran, Hao Wang, Zihe Song, Mengzhou Wu, Yuan Cao, Ying Zhang, Wei Yang, and Tao Xie. 2024. Guardian: A Runtime Framework for LLM-Based UI Exploration. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, Maria Christakis and Michael Pradel (Eds.). ACM, 958–970.

[39] Navid Salehnamadi, Forough Mehralian, and Sam Malek. 2022. Groundhog: An Automated Accessibility Crawler for Mobile Apps. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 50:1–50:12.

[40] Ben Shneiderman and Catherine Plaisant. 2010. *Designing the User Interface - Strategies for Effective Human-Computer Interaction, 5th Edition.* Addison-Wesley.

[41] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, stochastic model-based GUI testing of Android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 245–256.

[42] Yanqi Su, Dianshu Liao, Zhenchang Xing, Qing Huang, Mulong Xie, Qinghua Lu, and Xiwei Xu. 2024. Enhancing Exploratory Testing by Large Language Model and Knowledge Graph. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 98:1–98:12.

[43] Maryam Taeb, Amanda Swearngin, Eldon Schoop, Ruijia Cheng, Yue Jiang, and Jeffrey Nichols. 2024. AXNav: Replaying Accessibility Tests from Natural Language. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*, Florian 'Floyd' Mueller, Penny Kyburz, Julie R. Williamson, Corina Sas, Max L. Wilson, Phoebe O. Toups Dugas, and Irina Shklovski (Eds.). ACM, 962:1–962:16.

[44] W3C. 2022. Rule-based accessibility and age-friendly development framework. https://www.w3.org/2022/09/hangzhou/a11y/talk-xuhuaixin.html. Accessed: 2026.

[45] Chenxu Wang, Tianming Liu, Yanjie Zhao, Minghui Yang, and Haoyu Wang. 2025. LLMDroid: Enhancing Automated Mobile App GUI Testing Coverage with Large Language Model Guidance. *Proc. ACM Softw. Eng.* 2, FSE (2025), 1001–1022.

[46] Dingbang Wang, Yu Zhao, Sidong Feng, Zhaoxu Zhang, William G. J. Halfond, Chunyang Chen, Xiaoxia Sun, Jiangfan Shi, and Tingting Yu. 2024. Feedback-Driven Automated Whole Bug Report Reproduction for Android Apps. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, Maria Christakis and Michael Pradel (Eds.). ACM, 1048–1060.

[47] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-Agent: Autonomous Multi-Modal Mobile Device Agent with Visual Perception. *CoRR* abs/2401.16158 (2024). arXiv:2401.16158

[48] Xuan Wang, Yingchuan Wang, Yongxiang Hu, Yu Zhang, Hailiang Jin, Shiyu Guo, Juxing Yuan, and Yangfan Zhou. 2025. From Redundancy to Efficiency: Exploiting Shared UI Interactions towards Efficient LLM-Based Testing. In *ASE 2025 Industry Showcase, 40th IEEE/ACM International Conference on Automated Software Engineering (ASE 2025), Seoul, South Korea, November 16-20, 2025*. To appear.

[49] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.* 13, 4 (2004), 600–612.

[50] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*.

[51] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. AutoDroid: LLM-powered Task Automation in Android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, ACM MobiCom 2024, Washington D.C., DC, USA, November 18-22, 2024*, Weisong Shi, Deepak Ganesan, and Nicholas D. Lane (Eds.). ACM, 543–557.

[52] Yuxuan Yan, Zhenhua Li, Qi Alfred Chen, Christo Wilson, Tianyin Xu, Ennan Zhai, Yong Li, and Yunhao Liu. 2019. Understanding and Detecting Overlay-based Android Malware at Market Scales. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys 2019, Seoul, Republic of Korea, June 17-21, 2019*, Junehwa Song, Minkyong Kim, Nicholas D. Lane, Rajesh Krishna Balan, Fahim Kawsar, and Yunxin Liu (Eds.). ACM, 168–179.

[53] Bo Yang, Zhenchang Xing, Xin Xia, Chunyang Chen, Deheng Ye, and Shanping Li. 2021. Don't Do That! Hunting Down Visual Design Smells in Complex UIs against Design Guidelines. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 761–772.

[54] Bo Yang, Zhenchang Xing, Xin Xia, Chunyang Chen, Deheng Ye, and Shanping Li. 2021. UIS-Hunter: Detecting UI Design Smells in Android Apps. In *43rd IEEE/ACM International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 89–92.

[55] Wenwen Yu, Zhibo Yang, Jianqiang Wan, Sibo Song, Jun Tang, Wenqing Cheng, Yuliang Liu, and Xiang Bai. 2025. OmniParser V2: Structured-Points-of-Thought for Unified Visual Text Parsing and Its Generality to Multimodal Large Language Models. *CoRR* abs/2502.16161 (2025). arXiv:2502.16161

[56] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025. AppAgent: Multimodal Agents as Smartphone Users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, CHI 2025, YokohamaJapan, 26 April 2025- 1 May 2025*, Naomi Yamashita, Vanessa Evers, Koji Yatani, Sharon Xianghua Ding, Bongshin Lee, Marshini Chetty, and Phoebe O. Toups Dugas (Eds.). ACM, 70:1–70:20.

[57] Mengxi Zhang, Huaxiao Liu, Changhao Du, Tengmei Wang, Han Li, Pei Huang, and Chunyang Chen. 2025. Distinguishing GUI Component States for Blind Users Using Large Language Models. *ACM Trans. Softw. Eng. Methodol.* 34, 8 (2025), 222:1–222:35.

[58] Dehai Zhao, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Seenomaly: vision-based linting of GUI animation effects against design-don't guidelines. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 1286–1297.