UI-Lens: Assessing General MLLMs' Potential to Automate UI Display Quality Assurance

Supplementary Material

Appendix Summary

We provide all additional details for our paper in the following sections.

- **Appendix A (Metadata)** details the dataset composition, collection strategies, and the manual defect synthesis pipeline in Figma.
- **Appendix B (Annotation Guidelines)** provides detailed guidelines for annotating various types of UI defects, including definitions, constraints, and visual examples for each anomaly type.
- **Appendix C (Prompts)** presents the detailed prompt templates used in our UI defect detection evaluation framework for each type of UI defects.
- **Appendix D (Model Information)** summarizes the model sources and inference configurations of the multimodal large language models used in the UI-Lens benchmark.
- **Appendix E (Metrics)** details the evaluation metrics for both single-interface and sequence-interface display defect detection tasks.
- **Appendix F (Detailed Related Works)** provides a broader discussion of additional related works.

A. Metadata

This section details the metadata composition of the UI-Lens dataset, including key information such as collection devices, application scenarios, interface modes, and sequence design, ensuring the dataset's diversity, authenticity, and reproducibility.

A.1. Device Models

Data collection covers 21 mainstream mobile devices, including various form factors like smartphones, tablets, and foldable screens. Specific models are listed in Table 1.

A.2. Application Scenario Coverage

The dataset includes 11 mainstream Chinese commercial applications, covering core scenarios such as social networking, entertainment, content creation, e-commerce, and lifestyle services, as shown in Table 2.

A.3. Collection Modes

To evaluate model robustness under different display conditions, data collection covers the following modes:

- **Color Mode:** Light Mode, Dark Mode (collected based on actual app support)
- **Text Mode:** Standard Mode, Large Text Mode (collected based on actual app support)
- **Orientation:** Portrait, Landscape (for tablet devices)
- **Operating System:** iOS, iPadOS, Android, HarmonyOS

A.4. Interface Collection Strategy

- **Single-Interface Collection:** Typical user interfaces were selected from each application, covering core functionalities while avoiding high-risk interfaces involving user privacy.
- **Sequence-Interface Collection:** Complete interaction sequences were designed for each application to simulate real user operation paths, used for evaluating cross-interface semantic consistency (*e.g.*, shopping flows, group buying flows).
- **Privacy Handling:**
  1. Information sourced from the collectors' public user profiles was used with authorization.
  2. Public information from online users, such as avatars, nicknames, and works, was anonymized to prevent identification of specific individuals.
  3. All biometric information (*e.g.*, faces) and real identity information (*e.g.*, addresses) were redacted.

A.5. Interface Metadata Example

The following is an example of a typical interface metadata record, showing its structure:

```
{
  "application": "Meituan",
  "device": "apple-iPhone17,1",
  "interface": "15022",
  "mode": "normal"
}
```

Corresponding interface details:

- **Interface ID:** 15022
- **Scenario:** Lifestyle Service
- **Interface Name:** Purchase Notes
- **Operation Path:** Login → Click group buying icon → Select category (*e.g.*, Food/Entertainment/Beauty) → Scroll down → Select group buy item → Click Purchase Notes tab or label

A.6. Figma Synthesis Pipeline

To ensure the high fidelity and visual realism of our dataset, the synthesized defective images were manually generated by design experts via professional image manipulation in

Device Type	Specific Models
Smartphones	iPhone 16 Pro Max, iPhone SE, iPhone 13 mini, iPhone X, iPhone 16 Pro, iPhone 16
Tablets	iPad Air 2 (Landscape/Portrait), iPad 10.2 (Landscape/Portrait), Huawei MatePad Pro (Landscape/Portrait), Xiaomi Pad 6 Pro (Landscape/Portrait)
Foldable Devices	Huawei Mate x5 (Inner/Outer Screen)
Other Phones	Honor X10 Max, Xiaomi 13, Huawei P20, Huawei P60, Huawei Mate 40

Table 1. Device models covered in the dataset.

Scenario Category	Sub-category	Application Name
Social Entertainment	Short/Medium Video	Douyin, Xigua, Jingxuan
	News/Information Aggregation	Toutiao
	Novels/Short Dramas	Fanqie, Hongguo
	Lifestyle Sharing	Douyin, Kesong
Creation	Content Production Assistance	Douyin, Jianying, Jimeng
E-commerce	E-commerce Transactions (Sequential Interface)	Douyin, PDD
Lifestyle Services	Group Buying (Sequential Interface)	Douyin, Meituan

Table 2. Application scenarios covered in the dataset.

Figma (a collaborative UI design tool similar to Photoshop). Instead of relying on automated scripts that might introduce unnatural artifacts, manual construction guarantees that the injected defects mimic authentic rendering failures perfectly.

Figure 1 illustrates our step-by-step synthesis pipeline, using the construction of an "Abnormal Text Ellipsis" defect as an example:

- **Original Interface:** The process begins with a pristine, bug-free commercial UI screenshot (*e.g.*, a search bar displaying complete text).
- **Step 1: Masking Original Content.** The designer creates a solid-color rectangle whose fill exactly matches the local background color of the UI component. This shape is precisely positioned to cover the specific portion of the text intended to be truncated.
- **Step 2: Injecting the Defect.** A new text layer containing the synthetic defect—in this case, an ellipsis ("...")—is inserted over the masked area. The designer meticulously aligns the font family, font size, and text color with the original component to maintain strict visual consistency.
- **Synthetic Results:** The final output is a seamless, highly realistic defective UI image, ready to be incorporated into the benchmark.

B. Annotation Guidelines

This section provides detailed guidelines for annotating various types of UI defects. These guidelines are designed to ensure consistency and accuracy in the identification and classification of UI issues. Each subsection focuses on a specific anomaly, outlining its definition, constraints, and visual examples to aid annotators in their task.

B.1. Annotation Guidelines for Text Overflow

Text Overflow is a common UI defect where text extends beyond the boundaries of its designated container. This can happen when the text content is too long for the container and no proper handling (like truncation or wrapping) is in place. Figure 2 shows a typical example of Text Overflow.

Constraint conditions. The container of the text must be visually discernible on the screen (for example, through a border, background color, shadow, shape, or other visual cues that allow users to directly perceive its boundaries). If the container itself is not visually apparent, then the issue of text overflowing the container does not arise.

As detailed in Table 3, Text Overflow can be categorized into several sub-types, such as text overflowing its container, a combined text-and-icon unit breaking its bounds, or the entire text container extending beyond its parent.

B.2. Annotation Guidelines for Cropped Content

Cropped Content refers to a UI issue where a container or its content is partially cut off, leading to an incomplete presentation. This can be caused by improper layout calculations. An example of this issue is presented in Figure 3.

Constraint conditions:

1. Clipping of scrollable elements is not considered an issue.
2. Gradient effects are not considered an issue.

Figure 1. The manual image synthesis pipeline in Figma, illustrating the step-by-step construction of an “Abnormal Text Ellipsis” defect.

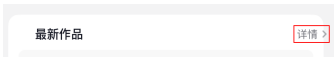
Type	Description	Example
Text out of bounds	Any text pixels that overflow beyond the boundaries of the current text container.	
Combined text-and-icon out of bounds	The icon and text form a single unit; if either the icon or the text overflows their shared container.	
The overall text container goes out of bounds	The overall container holding the text itself extends beyond its parent container.	

Table 3. Sub-types and examples of Text Overflow.

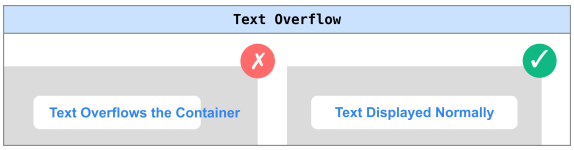


Figure 2. Example of Text Overflow.

3. User-generated content (such as text inherent to video content, user-generated comments, etc.) should not be considered.
4. Visually insignificant clipping is not considered an issue.

Table 4 illustrates the different forms of this defect, including occlusion truncation, where the content is truncated by the in-container spacing, and abrupt hard truncation, where text is cut off without an ellipsis.

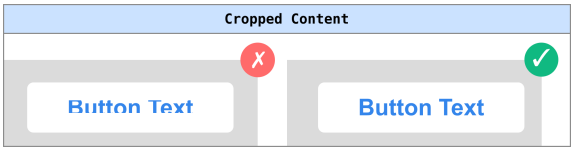


Figure 3. Example of Cropped Content.

B.3. Annotation Guidelines for Container Overlap

Container Overlap occurs when two or more UI containers are improperly rendered, causing them to overlap and obscure one another. This often results from incorrect positioning or sizing of elements. Figure 4 provides a clear example of this defect.

Constraint conditions:

1. Bullet comments overlapping with each other or with other components are not considered an issue.
2. Suspected element overlap in user-generated content (such as user-uploaded images or videos) is not considered an issue.
3. Overlap between the screen recording component and other elements is not considered an issue.
4. Overlapping components within frames used for gradient effects are not considered an issue.

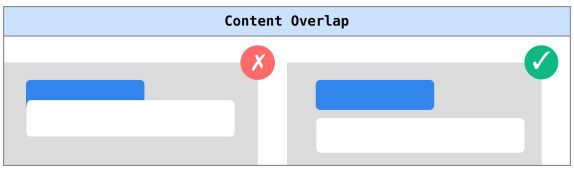


Figure 4. Example of Container Overlap.

The primary causes are detailed in Table 5, which include excessive text length forcing a container to overlap with others, and components simply overlapping due to layout miscalculations.

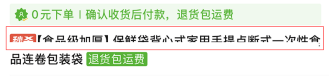
Type	Description	Example
Occlusion truncation	The content is truncated by another component, the screen edge, a container, etc., resulting in incomplete presentation of the information.	
Abrupt hard truncation	The text content is abruptly cut off, with no ellipsis and no residual characters.	

Table 4. Sub-types and examples of Cropped Content.

Type	Description	Example
Excessive text length	The text is too long, causing it to overlap with other content.	
Component overlap with other components	The entire component overlaps with other components.	

Table 5. Sub-types and examples of Container Overlap.

B.4. Annotation Guidelines for Undisplayed Content

An Undisplayed Content is a UI defect where a container that should display content appears empty or blank. This can be due to data failing to load, rendering errors, or incorrect component state management. A visual example is shown in Figure 5.

Constraint conditions:

1. Placeholder graphics (such as default avatars) and content in a loading state that has not yet been displayed are not considered issues.
2. On the creation page, it is not considered an issue if content that is pending upload or editing is empty.
3. Content that is not fully displayed due to insufficient page space is not considered an issue.

This defect can manifest in several ways, as described in Table 6. These include a single blank component, visually inconsistent cells in a repeated view, asymmetric whitespace, or abnormally large whitespace where content is missing.

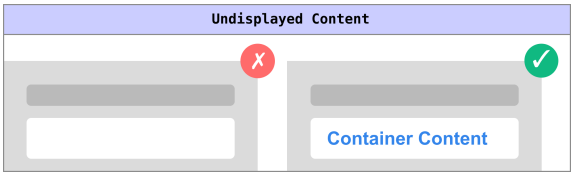


Figure 5. Example of Undisplayed Content.

B.5. Annotation Guidelines for Abnormal Text Ellipsis

Abnormal Text Ellipsis is a specific type of text-handling issue where an ellipsis ("...") is used inappropriately. While ellipses are often used to indicate truncated text, their application in certain contexts, such as on interactive buttons or very short text fields, can be considered a UI defect. Figure 6 illustrates an instance of this issue. Table 7 lists four issue sub-types: (1) truncated buttons/tags, (2) ultra-short text, (3) verb-ending prompts, (4) text–icon blocks with leftover space.

Figure 6. Example of Abnormal Text Ellipsis.

B.6. Annotation Guidelines for Text Inconsistency

Text Inconsistency arises when information on a UI is contradictory, ambiguous, or presents information inconsistently across different parts of the application. This can confuse users and damage credibility. Figure 7 shows an example of such an inconsistency.

The various forms of this defect are outlined in Table 8. These include conflicts in numbers or wording, loss of key information between screens, and the sudden appearance of undisclosed information at critical points in a user flow.

Type	Description	Example
Single blank component	An independent component with clearly defined functional boundaries that is completely or partially blank (gray/white) inside.	
Visually inconsistent cell in repeated view	In a view containing multiple repeated structural units, one or more cells visually differ significantly from surrounding cells that have loaded normally.	
Asymmetric whitespace	There is unnatural blank space inside an asymmetric component (usually on the left side of the text).	
Abnormally large whitespace	There is a title component with space below it, but the actual content below is missing.	

Table 6. Sub-types and examples of Undisplayed Content.

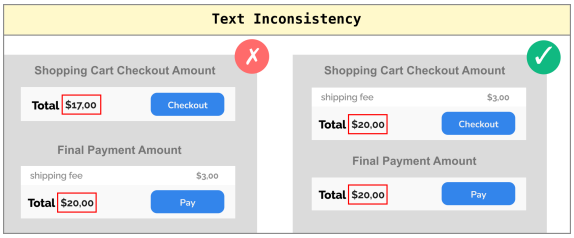


Figure 7. Example of Text Inconsistency.

C. Prompts

This section presents the detailed prompt templates used in our UI defect detection evaluation framework. Each prompt is designed to guide MLLMs in detecting specific types of UI defects. For ease of reading, we provide English translations of the prompts here; note that the actual evaluations on the Chinese dataset were conducted using their original Chinese versions.

C.1. Prompts for Text Overflow

Role Definition: You are a **Mobile UI Quality Assurance Assistant**. Your task is to determine whether there is **Text Overflow** in the interface.

Text Overflow Detection Guidelines:

Problem Constraints:

- Detection is limited to **visually distinct containers** (background areas with clues like borders, background colors, shadows, or shapes).
- If text is not within a clearly identifiable container (*e.g.*, transparent background, no surrounding frame), it should

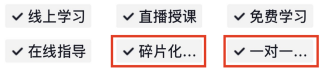
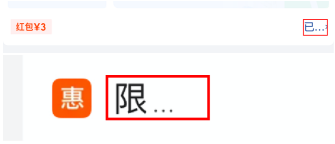
Type	Description	Example
Component text	Text in interactive components (*e.g.*, button components) and labeling components (*e.g.*, tag components) is truncated with an ellipsis.	
Overly short text	When the Chinese character count is ≤ 2, the English character count is ≤ 4, or the digit character count is ≤ 4, the presence of an ellipsis is regarded as an omission anomaly.	
Incomplete semantics	System prompt text* contains an ellipsis, resulting in incomplete semantics (ending with a verb). *Note**: "System prompt text" refers to short sentences that are automatically generated by a system or app-wide framework to provide users with status updates, results, or warning messages.	
Abnormal blank space	A text-only element, or a combination of text and icons, in which the text contains an ellipsis and a large blank space still remains after the text or icons.	

Table 7. Sub-types and examples of Abnormal Text Ellipsis.

be skipped and not considered an issue.

- Judgment of boundary crossing should be based on the actual visual presentation in the image, ignoring minor blurring from compression or anti-aliasing.
- Recognition of text content does not need to be perfectly accurate; approximate inference can be made based on position and shape.

Types of Text Overflow Issues to Report:

1. **Text Crossing Boundary** — Any character's pixels clearly exceed its container's boundary.
2. **Text-Icon Combination Out of Bounds** — An icon and text form a single visual group (like a button or tab), and either part overflows their common container.
3. **Container Out of Bounds** — A container that includes text overflows its parent container's area.

Do Not Report:

- Image watermarks, such as "Douyin ID: xxxx" or "CCTV5" on a background image.
- Text embedded within video content.
- Text obscured by a screen recording icon.

Reasoning Process:

1. **Observation & Analysis** — Analyze interface structure and elements.
2. **Element Extraction** — Extract text with background colors and UI element component information from the image.
3. **Guideline Judgment** — Identify candidate text with issues based on the detection guidelines.
4. **Semantic Filtering** — Judge and filter out non-UI functional text from the candidates.
5. **Issue Output** — Output a standard JSON report in the specified format.

Output Format:

```
{
  "reasoning_process": "<Brief: Element Analysis
    ->Text and Component Extraction->Abnormality
    Judgment->Non-UI Functional Text Filtering->
    Result Judgment>",
  "total_text_overflow_issues": <integer, 0 for
    no issues>,
  "issue_descriptions": [
    {
      "text_overflow_description": "<[Text
    Content] has a text overflow issue, reason: [
    Explain based on the issue type and whether
    it is non-UI functional text]>",
      "location": "<Top/Middle/Bottom + Left/
    Center/Right>"
    }
  ]
```

Type	Description	Example
Inconsistent numbers	Numerical conflict. The numerical information displayed for a product across different channels or pages is inconsistent (including discrepancies in time validity, price, quantity, geographic distance, etc.). *Note**: For the same destination, a geographic distance difference greater than 10 meters is considered inconsistent.	• 58.9 VS 51.6. • Advertised as 25% cheaper than competitors, but the actual discount does not reflect this claim.
	Specification conflict. Product specification information shows inconsistencies across different display locations, such as non-standardized units, coexistence of vague and precise descriptions, etc.	• 500 g VS 0.5 kg (unit conflict). • 530,000 VS 500,000+ (vague description conflicts with precise description).
Inconsistent wording	The same meaning is expressed using different wording.	• *Valid only Monday to Friday* VS *Unavailable dates: Saturday, Sunday*.
	Self-contradictory text	• Free shipping VS Shipping discount.
	Ambiguous text	*2 for $5* could mean: • 2 items for $5 total. • 2 of something (like 2 pounds, 2 dozen) for $5.
Lost information	Key information is lost upon screen navigation within the user flow, complicating the user's decision-making.	• The promotional information (*e.g.*, Save ¥20) displayed on the product list page is no longer shown on the product detail page.
Additional information	The sudden introduction of previously undisclosed critical information at key conversion points in the user journey, such as the checkout or payment pages, can influence user decisions and may even foster a sense of deception.	• Extra services are only disclosed at the final payment stage, with no prior indication earlier in the user journey.

Table 8. Sub-types and examples of Text Inconsistency.

```
}
```

C.2. Prompts for Cropped Content

Role Definition: You are a **Mobile UI Quality Assurance Assistant**. Your task is to determine whether there is **Cropped Content** in the interface.

Cropped Content Detection Guidelines:

Types of Cropped Content Issues to Report:

1. **Element Edge Clipping** — Icons or components are truncated, or text content and icon information are not fully displayed. (Exception: truncation caused by proximity to screen edges).
2. **Sudden Text Truncation** — Text is truncated horizontally or vertically without an ellipsis ("...") or gradient effects, resulting in incomplete strokes.

Do Not Report:

1. Element clipping within scrollable areas (*e.g.*, lists, scroll containers).
2. Elements that gradually fade out.
3. User-generated content (*e.g.*, text within videos, user-generated comments).

4. Minor clipping that is not visually obvious.

Reasoning Process:

1. **Observation & Identification** — Analyze the interface structure and classify elements (*e.g.*, text, icons, components, containers).
2. **Focus Inspection** — Examine screen edges, component boundaries, container edges, and text endings.
3. **Clipping Judgment** — Identify whether elements are truncated, strokes or icons are incomplete, or text is abruptly cut off.
4. **Guidelines Validation** — Exclude scrollable areas, fading elements, User-generated content, and non-obvious clipping.
5. **Exception Validation** — Confirm whether text truncation is intentional (*e.g.*, for product titles).
6. **Issue Confirmation** — Record clipping types and locations; if no issues, record the "total number of element clipping issues" as 0.
7. **Issue Output** — Generate a standard JSON report in the specified output format.

Output Format:

```
{
  "reasoning_process": "<Brief: Observation->
    Focus Inspection->Clipping Judgment->
    Guidelines Validation->Exception Validation>"
    ,
  "total_element_clipping_issues": <integer>,
  "issue_descriptions": [
    {
      "cropped_element_description": "<[Element
    Type] is cropped at [truncation position],
    reason:[element clipping issue type]>",
      "location": "<Top/Middle/Bottom + Left/
    Center/Right>"
    }
  ]
}
```

C.3. Prompts for Container Overlap

Role Definition: You are a **Mobile UI Quality Assurance Assistant**. Your task is to determine whether there is **Container Overlap** in the interface.

Container Overlap Detection Guidelines:

Types of Container Overlap Issues to Report:

1. **Text Overlap** — Text in titles, labels, or buttons is too long, causing it to obscure other component elements.
2. **Component Overflow Overlap** — Components like cards, lists, or pop-ups are displaced, causing them to obscure other components.

Do Not Report:

1. Any overlap related to bullet comments (danmaku).
2. Content overlap within video players.
3. Overlap in thumbnails or previews.
4. Overlap related to screen recording components.
5. Overlap caused by toasts or pop-ups.

Reasoning Process:

1. **Observation & Identification** — Analyze the interface structure and classify elements (containers/interactive/-text/images).
2. **Focus Inspection** — Examine top/bottom junctions, floating buttons, text-dense areas, etc.
3. **Boundary Judgment** — Identify element boundary overlap, determine layer relationships and the degree of obstruction.
4. **Guidelines Validation** — Exclude content that should not be reported.
5. **Issue Confirmation** — Record element types, content, and locations; if no issues, record "total number of container overlap issues" as 0.
6. **Issue Output** — Output a standard JSON report according to the output format.

Output Format:

```
{
  "reasoning_process": "<Brief: Observation &
    Identification->Focus Inspection->Boundary
    Judgment->Guidelines Validation>",
  "total_container_overlap_issues": <integer>,
  "issue_descriptions": [
    {
      "overlapping_element_description": "<[
    Element A] overlaps with [Element B], reason
    :[container overlap issue type]>",
      "location": "<Top/Middle/Bottom + Left/
    Center/Right>"
    }
  ]
}
```

C.4. Prompts for Undisplayed Content

Role Definition: You are a **Mobile UI Quality Assurance Assistant**. Your task is to determine whether there is **Undisplayed Content** in the interface.

Undisplayed Content Detection Guidelines:

Types of Undisplayed Content Issues to Report:

1. **Key Component Blank** — Independent functional components are completely/partially blank (gray/white)
2. **List Element Anomaly** — A cell in a list/grid shows significant differences from surrounding loaded cells
3. **Asymmetric Whitespace** — Left and right internal padding difference of elements (*e.g.*, cards) $>$ 8px
4. **Semantic Content Missing** — Title component exists with space below, but actual content is missing

Do Not Report:

1. Has placeholder images (default avatars, broken image icons) or loading states (loading indicators, skeleton screens)
2. User-generated content production pages where content to be uploaded is empty / chat history is empty
3. Content missing or partially displayed due to insufficient page space

Reasoning Process:

1. **Observation & Identification** — Interface structure, component classification (cards/lists/titles/images, etc.)
2. **Focus Inspection** — Key component areas, list/grid cells, areas below titles
3. **Blank Area Judgment** — Identify blank areas, determine if functional components are missing, list anomalies, asymmetric whitespace
4. **Semantic Reasoning** — Analyze whether content should exist below the title but is actually missing
5. **Guidelines Validation** — Exclude placeholder images, loading states, empty User-generated content states, insufficient space situations
6. **Issue Confirmation** — Record missing types and locations; if no issues, record "total number of content missing issues" as 0
7. **Issue Output** — Output standard JSON report according to the output format

Output Format:

```
{
  "reasoning_process": "<Brief: Observation->
    Inspection->Judgment->Reasoning->Validation>"
    ,
  "total_content_missing_issues": <integer>,
  "issue_descriptions": [
    {
      "missing_content_description": "<[Component
    /Area] has [missing type], reason:[content
    missing issue type]>",
      "location": "<Top/Middle/Bottom + Left/
    Center/Right>"
    }
  ]
}
```

C.5. Prompts for Abnormal Text Ellipsis

Role Definition: You are a **Mobile UI Quality Assurance Assistant**. Your task is to determine whether there is **Abnormal Text Ellipsis** in the interface.

Abnormal Text Ellipsis Detection Guidelines:

Basic Concepts:

- **System Prompt Text**: System/App-generated hint messages (*e.g.*, placeholders, page titles).
- **Text Separator**: Text before and after "—" are two independent texts.

Types of Abnormal Text Ellipsis Issues to Report:

1. **Interactive Component Text Ellipsis** — Text in a Button component shows an ellipsis.
2. **Tag Component Text Ellipsis** — Text in a Tag component shows an ellipsis.
3. **Overly Short Text Ellipsis** — Text with an ellipsis is too short (*e.g.*, $\leq$ 2 Chinese characters, $\leq$ 4 English characters, or $\leq$ 4 digits).
4. **Incomplete Semantics in System Prompt Text** — System prompt text has an ellipsis and ends with a verb, resulting in incomplete semantics.
5. **Whitespace to the Right of Body Text** — Normal body text has an ellipsis, but there is significant whitespace to its right.
6. **Whitespace in Text-Icon Combination** — A combination of text and an icon has an ellipsis in the text, but there is significant whitespace at the end.

Do Not Report:

- Normal ellipsis for long text without excess whitespace.
- Ellipsis with clear design intent (*e.g.*, "More...").

Reasoning Process:

1. **Identify Text** — Classify text: Interactive Component (Button), Tag Component (Tag), System Prompt Text, Normal Body Text, or Text-Icon Combination.
2. **Locate Ellipsis** — Find all text containing "..." or "...".
3. **Item-by-Item Check** — For each text with an ellipsis, determine:
   - Is it a Button/Tag component?
   - Is the character count $\leq$ the threshold (2 for Chinese, 4 for English, 4 for digits)?
   - Is it system prompt text with incomplete semantics?
   - Is there whitespace to the right? (For body text or text-icon combinations).
4. **Guidelines Validation** — Report if it meets any of the 6 abnormal conditions.
5. **Issue Confirmation** — Record the type, content, and location; if no issues, record "total number of abnormal text ellipsis issues" as 0.
6. **Issue Output** — Output a standard JSON report according to the output format.

Output Format:

```
{
  "reasoning_process": "<Brief: Identify Text->
    Locate Ellipsis->Item-by-Item Check->
    Guidelines Validation>",
  "total_abnormal_text_ellipsis_issues": <integer
    >,
  "issue_descriptions": [
    {
      "abnormal_ellipsis_description": "<[Text
    Content] has an abnormal ellipsis, reason:[
    abnormal text ellipsis issue type]>",
```

```
      "location": "<Top/Middle/Bottom + Left/
    Center/Right>"
    }
  ]
}
```

C.6. Prompts for Text Inconsistency

Information Extraction Prompt

Role Definition: You are a professional UI description expert, skilled at accurately and objectively describing interface content. Your task is to extract and clearly describe all visible information from the provided interface screenshot.

Core Task:

- Identify the overall layout and main sections of the interface.
- Describe all visible elements (text, buttons, images, icons, etc.) in each section.
- Note visual features such as text content, color, and size.

Output Requirements:

- Use clear, objective language.
- Describe elements from top to bottom, left to right.
- Maintain coherence and hierarchy.
- Only describe visible content.
- Clearly describe important business information.

Information Analysis Prompt

Role Definition: You are a professional information consistency analysis expert, specializing in reviewing information consistency issues in user browsing/testing flows.

Analysis Task:

1. **User Perspective:** Evaluate the impact of issues from a user experience perspective.
2. **Immediate Task:** Check for explicit contradictions.
3. **Hidden Information Inference:** Do not assume issues with hidden information.
4. **Language:** Use Chinese for reporting issues.

Output Format Requirements: Output the analysis results in the following JSON format.

```
[
    {
        "id": "output_01",
        "reasoning": "...(Please output the
    complete reasoning process)",
        "pages": [...(Output the pages involved
    in the problem, please strictly use the
    naming format starting from 01, such as "01",
     "02"...)],
        "details": "...(Explain the problem in as
     much detail as possible, specific content)"
    },...
]
```

Information Filtering and Merging Prompt

Role Definition: You are a professional information reviewer, responsible for classifying the preliminary analysis results of the information consistency analyst.

Task Requirements: Classify and tag each issue with a "detection_type" from the following:

- **Inconsistent numbers**: Numerical conflict, Specification conflict.
- **Inconsistent wording**: Same meaning with different wording, Self-contradictory text, Ambiguous text.
- **Lost information**: Key information is lost.
- **Additional information**: Sudden introduction of undisclosed critical information.

Output Format Requirements: Please strictly follow the JSON format below to output the processed results.

```
[
    {
        "id": "output_01 (please rename according
     to the output)",
        "detection_type": "Inconsistent numbers",
        "pages": [...(Keep the original content
    without any modification)],
        "details": "...(Keep the original content
     without any modification)"
    },...
]
```

D. Model Information

This section summarizes the model sources and inference configurations of the multimodal large language/vision-language models (MLLMs/VLMs) used in the UI-Lens benchmark, with the goal of facilitating reproducibility and fair comparison.

We evaluate a total of **9 widely used models**, including **7 closed-source** systems (Seed/Doubao, GPT, and Gemini families) and **2 open-source** systems (Qwen and GLM families). All models are evaluated under a unified **zero-shot** setting with consistent prompt templates and decoding strategies. Each configuration is run twice, and we report the average performance across the two runs.

D.1. Model Details and Sources

The specific model variants, release dates, and official documentation links are listed in Table 9. All models are accessed through their publicly available APIs or SDKs.

D.2. Experimental Settings and Hyperparameters

To ensure fairness and comparability across different providers, we adopt a unified inference configuration whenever supported by the corresponding APIs. Table 10 summarizes the generation parameters used for each model.

Model Name	Version/Date	Source
Seed1.6	202506	https://www.volcengine.com/docs/82379/1593702
Seed1.6-Vision	202508	https://www.volcengine.com/docs/82379/1799865
Seed1.5-VL	202504	https://www.volcengine.com/docs/82379/1554521
GPT-5	202508	https://platform.openai.com/docs/models/gpt-5
GPT-4.1	202504	https://platform.openai.com/docs/models/gpt-4.1
GPT-4o	202411	https://platform.openai.com/docs/models/gpt-4o
Gemini-2.5-Pro	202501	https://ai.google.dev/gemini-api/docs/models#gemini-2.5-pro
Qwen3-VL	202507	https://modelscope.cn/models/Qwen/Qwen3-VL-235B-A22B
GLM-4.5V	202508	https://docs.zhipu.ai/docs/glm-4-5v

Table 9. Evaluated models with corresponding versions and official sources.

Model	Generation Setup	QPM/TPM
Seed1.6	`{"temperature": 0.1, "top_p": 0.1, "max_tokens": 8192}`	120/600,000
Seed1.6-Vision	`{"temperature": 0.1, "top_p": 0.1, "max_tokens": 8192}`	120/600,000
Seed1.5-VL	`{"temperature": 0.1, "top_p": 0.1, "max_tokens": 8192}`	120/600,000
GPT-5	`{"temperature": 0.1, "top_p": 0.2, "max_tokens": 8192}`	120/600,000
GPT-4.1	`{"temperature": 0.1, "top_p": 0.2, "max_tokens": 8192}`	120/600,000
GPT-4o	`{"temperature": 0.1, "top_p": 0.2, "max_tokens": 8192}`	120/600,000
Gemini-2.5-Pro	`{"temperature": 0.1, "top_p": 0.2, "max_tokens": 8192}`	120/600,000
Qwen3-VL	`{"temperature": 0.1, "top_p": 0.2, "max_tokens": 8192}`	120/600,000
GLM-4.5V	`{"temperature": 0.1, "top_p": 0.2, "max_tokens": 8192}`	120/600,000

Table 10. Generation parameters used for all evaluated models. QPM: queries per minute; TPM: tokens per minute.

Unified API Invocation To minimize confounding factors introduced by different serving environments, we standardize the invocation pipeline as follows:

- **Invocation protocol:** All models are accessed via MaaS (Model-as-a-Service) APIs provided by their respective platforms.
- **Temperature:** Fixed to 0.1 for all models to reduce randomness and improve the stability of zero-shot evaluation.
- **Max output tokens:** Set to 8192 for image-based dialogue responses, ensuring that models are not artificially truncated on longer UI understanding tasks.
- **Sampling strategy:** We use nucleus sampling with top-p in the range 0.1–0.2, depending on model-specific recommendations. This keeps generation focused while maintaining a controlled degree of diversity.
- **Concurrency:** For each model, the per-model `max_concurrency` is set to 20, so that latency and rate-limit behavior are aligned during batch evaluation.
- **Experiment management:** We use LangSmith[1] to manage datasets, track experiment configurations, and aggregate evaluation metrics, enabling systematic logging and reproducibility.

[1] https://smith.langchain.com/

E. Metrics

This section details the metrics used to evaluate the performance of MLLMs on UI defect detection tasks. The evaluation is divided into two distinct scenarios: single-interface display defect detection and sequence-interface display defect detection.

E.1. Single-Interface Display Defect Detection

For single interface defect detection, the task is formulated as a binary classification problem. Given a single UI screenshot, the model must determine whether a specific type of UI defects is present.

The evaluation metrics used for this task are **Accuracy**, **Precision**, **Recall**, and **F1 Score**. These metrics are calculated based on the following definitions:

- **True Positive (TP)**: The model correctly identifies a UI screenshot that contains a defect.
- **True Negative (TN)**: The model correctly identifies a UI screenshot that does not contain a defect.
- **False Positive (FP)**: The model incorrectly identifies a defect in a UI screenshot that is actually defect-free.
- **False Negative (FN)**: The model fails to identify a defect in a UI screenshot that actually contains one.

The formulas for the metrics are as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Note: This evaluation framework includes all display defect types with the exception of text inconsistency, which is evaluated under the multi-UI defect detection scenario.

E.2. Sequence-Interface Display Defect Detection

For defects that require context from sequential screenshots, such as text inconsistencies across a user flow, the task is treated as an open-ended recall problem.

The model's output is a set of identified defects, where each defect includes a detailed description and the set of pages on which it occurs. This output set is compared against a manually curated groundtruth answer set. The metrics used are **Precision**, **Recall**, and **F1 Score**.

A predicted defect is considered a **True Positive (TP)** if it meets the following two conditions:

1. **Page Intersection**: The set of pages associated with the predicted defect has a non-empty intersection with the pages of a groundtruth defect.
2. **Details Consistency**: The description of the predicted defect is semantically consistent with the description of the groundtruth defect.

Based on this matching process, the metrics are calculated as follows:

- **True Positives (TP)**: The number of predicted defects that successfully match a groundtruth defect.
- **False Positives (FP)**: The number of predicted defects that do not match any groundtruth defects.
- **False Negatives (FN)**: The number of groundtruth defects that are not matched by any predicted defects.

The formulas for the metrics are:

$$\text{Precision} = \frac{|TP|}{|TP| + |FP|} = \frac{|\text{Matched Predictions}|}{|\text{All Predictions}|}$$

$$\text{Recall} = \frac{|TP|}{|TP| + |FN|} = \frac{|\text{Matched Predictions}|}{|\text{All Groundtruth Defects}|}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F. Detailed Related Works

In recent years, multimodal large language models (MLLMs) and vision–language models (VLMs) have achieved significant advances in user interface understanding [5, 6, 14, 16]. Studies show that even general-purpose foundation models (*e.g.* Gemini-2.5-Pro) demonstrate strong visual–structural parsing and semantic reasoning capabilities: they can accurately recognize UI components, interpret textual content, and infer interface functionality [2], thereby laying the groundwork for more complex UI tasks.

Meanwhile, research interest in leveraging multimodal visual reasoning for UI quality assurance has been steadily increasing [1, 4, 8]. Early systems such as OwlEye [9] and Nighthawk [10], built on visual understanding methods, can localize display defects in mobile applications, including element misalignment and occlusion. Recent works related to UI quality assurance further include: (1) end-to-end model-based approaches that directly detect UI defects, exemplified by AutoConsis [3], which combines multimodal models with large language models (LLMs) to detect data inconsistencies across application interfaces; (2) using MLLMs as judges to generate UI feedback aligned with human aesthetic and user-experience (UX) ratings [11–13, 15]. On the benchmarking side, WebRSSBench [7] evaluates MLLMs' color robustness and safety-critical detection capabilities on web interfaces, while GUI Testing Arena [17] assesses the ability of Graphical UI (GUI) agents to discover UI defects during execution.

However, as compared in Table 11, existing UI defects datasets typically suffer from an overreliance on algorithmically synthesized data or a restriction to single-interface analysis. To address these gaps, our proposed UI-Lens introduces 6 fine-grained defects categories—encompassing both visual anomalies and semantic understanding defect—featuring high-fidelity manual synthesis. Furthermore, it uniquely extends the evaluation to encompass both Single-UI and Sequential-UI scenarios.

References

[1] Ayoung Choi and Jongwook Jeong. Can LLMs Detect Display Issues? Uncovering the Impact of Prompting Techniques. In *Adjunct Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*, pages 1–3, 2025. 12

[2] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. 12

[3] Yongxiang Hu, Hailiang Jin, Xuan Wang, Jiazhen Gu, Shiyu Guo, Chaoyi Chen, Xin Wang, and Yangfan Zhou. AutoConsis: Automatic GUI-driven Data Inconsistency Detection of Mobile Apps. In *Proceedings of the 46th International*

Works	# Test set	# Types	Defect Categories	Image Source	Defect Origin	Context
OwlEye [9] (2021)	1,600	5	Component occlusion, Text overlap, Missing image, NULL value, Blurred screen	Crowdsourcing Public datasets	Real Algorithmic synthesis	Single-UI
Nighthawk [10] (2022)	9,600	4	Component occlusion, Text overlap, Missing image, NULL value	Crowdsourcing Public datasets	Real Algorithmic synthesis	Single-UI
AutoConsis [3] (2024)	120	1	Text inconsistency	–	Real Manual synthesis	Sequential-UI
UIClip [15] (2024)	201	9	Complexity, Color/Font Noise, Background Color, Spacing, Color/Font Swap, Text Contrast, Layout	Public datasets Generative models	Real	Single-UI
UI-Lens (Ours)	4,759 (zh) + 3,392 (en)	**6**	Text Overflow, Cropped Content, Container Overlap, Undisplayed Content, **Abnormal Text Ellipsis, Text Inconsistency**	**Automated** **Manual collection**	**Real** **Manual synthesis**	**Single-UI** **& Sequential-UI**

Table 11. Comparison of UI-Lens with existing UI defect datasets. UI-Lens differs by: (i) incorporating semantic understanding-related defects; (ii) introducing manually synthesized high-fidelity anomalies alongside real-world bugs; and (iii) extending the evaluation context from isolated static images (Single-UI) to dynamic, cross-interface sequences (Sequential-UI).

Conference on Software Engineering: Software Engineering in Practice, page 137–146, New York, NY, USA, 2024. Association for Computing Machinery. 12, 13

[4] Yongxiang Hu, Xuan Wang, Yingchuan Wang, Yu Zhang, Shiyu Guo, Chaoyi Chen, Xin Wang, and Yangfan Zhou. Auitestagent: Automatic requirements oriented gui function testing. *arXiv preprint arXiv:2407.09018*, 2024. 12

[5] Hongxin Li, Jingfan Chen, Jingran Su, Yuntao Chen, Li Qing, and Zhaoxiang Zhang. AutoGUI: Scaling GUI grounding with automatic functionality annotations from LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10323–10358, Vienna, Austria, 2025. Association for Computational Linguistics. 12

[6] Zhangheng Li, Keen You, Haotian Zhang, Di Feng, Harsh Agrawal, Xiujun Li, Mohana Prasad Sathya Moorthy, Jeff Nichols, Yinfei Yang, and Zhe Gan. Ferret-UI 2: Mastering universal user interface understanding across platforms. *arXiv preprint arXiv:2410.18967*, 2024. 12

[7] Junliang Liu, Jingyu Xiao, Wenxin Tang, Wenxuan Wang, Zhixian Wang, Minrui Zhang, and Shuanghe Yu. Benchmarking mllm-based web understanding: Reasoning, robustness and safety. *arXiv preprint arXiv:2509.21782*, 2025. UI feedback. 12

[8] Wei Liu, Feng Lin, Linqiang Guo, Tse-Hsun Peter Chen, and Ahmed E Hassan. Guiwatcher: Automatically detecting gui lags by analyzing mobile application screencasts. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 46–56. IEEE, 2025. 12

[9] Zhe Liu, Chunyang Chen, Junjie Wang, Yuekai Huang, Jun Hu, and Qing Wang. Owl eyes: spotting UI display issues via visual understanding. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, page 398–409, New York, NY, USA, 2021. Association for Computing Machinery. 12, 13

[10] Zhe Liu, Chunyang Chen, Junjie Wang, Yuekai Huang, Jun Hu, and Qing Wang. Nighthawk: Fully automated localizing UI display issues via visual understanding. *IEEE Transactions on Software Engineering*, 49(1):403–418, 2022. 12, 13

[11] Reuben A Luera, Ryan Rossi, Franck Dernoncourt, Samyadeep Basu, Sungchul Kim, Subhojyoti Mukherjee, Puneet Mathur, Ruiyi Zhang, Jihyung Kil, Nedim Lipka, et al. MLLM as a UI judge: Benchmarking multimodal LLMs for predicting human perception of user interfaces. *arXiv preprint arXiv:2510.08783*, 2025. 12

[12] Seokhyeon Park, Yumin Song, Soohyun Lee, Jaeyoung Kim, and Jinwook Seo. Leveraging multimodal llm for inspirational user interface search. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–22, 2025.

[13] Yuekai Wang and Franceska Xhakaj. AIHeurEval: Generating Heuristic Evaluations on Multiple UI Screens with Multimodal Large Language Models. In *International Conference on Human-Computer Interaction*, pages 228–237. Springer, 2025. 12

[14] Ziwei Wang, Weizhi Chen, Leyang Yang, Sheng Zhou, Shengchu Zhao, Hanbei Zhan, Jiongchao Jin, Liangcheng Li, Zirui Shao, and Jiajun Bu. MP-GUI: Modality Perception with MLLMs for GUI Understanding. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 29711–29721, 2025. 12

[15] Jason Wu, Yi-Hao Peng, Xin Yue Amanda Li, Amanda Swearngin, Jeffrey P Bigham, and Jeffrey Nichols. UIClip: a data-driven model for assessing user interface design. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pages 1–16, 2024. 12, 13

[16] Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe

Gan. Ferret-UI: Grounded mobile UI understanding with multimodal LLMs. In *European Conference on Computer Vision*, pages 240–255. Springer, 2024. 12

[17] Kangjia Zhao, Jiahui Song, Leigang Sha, Haozhan Shen, Zhi Chen, Tiancheng Zhao, Xiubo Liang, and Jianwei Yin. GUI Testing Arena: A unified benchmark for advancing autonomous gui testing agent. *arXiv preprint arXiv:2412.18426*, 2024. 12