# UI2APP: BENCHMARKING VISUAL INTERACTION INFERENCE IN EXECUTABLE WEB APPLICATION GENERATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Large language models (LLMs) have demonstrated growing competence in web page generation. However, existing text-driven approaches rely on complex prompts that impose substantial demands on users and offer limited expressivity for page layout and cross-page visual coherence. Image-driven paradigms, which take UI screenshots as input, align more closely with real development workflows. However, existing visual-reconstruction benchmarks focus primarily on visual fidelity and lack a systematic evaluation of the interaction capabilities in generated artifacts. To address this gap, we introduce UI2App, the first benchmark targeting *interaction inference*, the ability to infer application behavior from screenshots alone, without added behavioral guidance. UI2App comprises $600$ screenshots grouped into $95$ state-coherent screenshot sets for runnable multi-route web applications. We design an end-to-end pipeline that evaluates each artifact along three dimensions: executability, visual fidelity, and interaction inference. The interaction metric (IIS) assesses inferred interactions by functional correctness and state-management complexity, crediting any valid implementation rather than matching a single reference. Experiments on six frontier vision-language models reveal a marked capability mismatch between visual reconstruction and interaction realization: the visual-fidelity leader scores only $8.1$ on IIS, ranking fourth and trailing the IIS leader by $4.5\times$. High-complexity interactions such as cross-page state remain a pervasive bottleneck, with five of the six evaluated models scoring at most $3.5$ on this dimension. Overall, the results indicate that inferring complete interaction behavior from static screenshots remains a key challenge for models.

## 1 INTRODUCTION

Large language models (LLMs) have made notable progress in code generation, task planning, and reasoning (Roziere et al., 2023; Guo et al., 2024; 2025). Building on these advances, recent work has demonstrated the feasibility of end-to-end software development from natural-language instructions (Jimenez et al., 2024; Hong et al., 2024; Qian et al., 2024). However, existing text-driven paradigms largely rely on carefully designed or highly detailed prompts as input (Lu et al., 2025; Zhang et al., 2025), which introduces practical limitations. First, textual descriptions are limited in precisely specifying detailed visual requirements, making it difficult to accurately define page layouts and maintain visual coherence across multiple pages. Second, functional requirements involving cross-page interactions, such as state management and data synchronisation, are difficult to specify precisely and consistently in natural language.

To alleviate these limitations, image-to-webpage generation has emerged as an alternative paradigm. Instead of relying on textual specifications, this approach takes UI designs as input and reconstructs webpage structure and appearance directly from visual signals. This paradigm better aligns with the common designer workflow of turning mockups into functional prototypes. It also addresses the demand from non-technical users who want functional apps directly from screenshots. Several benchmarks have been developed along this line (Beltramelli, 2018; Si et al., 2025; Laurençon et al., 2024; Li et al., 2025).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.




Figure 1: **UI2App task overview: visual fidelity does not guarantee interaction-inference capability.** Same e-commerce screenshots, same action sequence (Add to Cart twice, then navigate to `/cart`). Claude Sonnet 4.6 reconstructs a fully dynamic application (cart counter increments, cross-route state propagates); Gemini 3.1 Pro Preview produces a higher-fidelity but *frozen* façade in which every interaction is a no-op. Only IIS catches this dichotomy.

However, these visual-reconstruction benchmarks largely focus on visual fidelity. They measure only how closely the rendered output matches a reference image, not whether the artifact *works* as an application. A generated interface may appear correct yet remain a behaviorally inert façade. As illustrated in Figure 1, different models might both produce runnable applications with high visual fidelity and basic navigation capabilities. However, while one model merely reconstructs static layouts, the other accurately infers and realizes the underlying dynamic interactions purely from static screenshots.

This gap reflects a deeper distinction in capability. When interaction behavior is supplied explicitly, through textual instructions or demonstration videos (He et al., 2026; Chen et al., 2025; Dai et al., 2026), the model performs *specification-following*: implementing a target that has already been named. When the input is screenshots alone, the model must perform *interaction inference*: recovering the missing behavior from visual evidence. No existing benchmark measures interaction inference under image-only input.

We introduce UI2App, a benchmark that makes interaction inference measurable. Each task supplies a set of screenshots and asks the model to emit a runnable multi-page artifact. Screenshots arrive without added behavioral guidance. The model must therefore interpret UI structure and semantics, infer the implied interactions, and implement a runnable framework-based application with shared components, routing, and coordinated state management across pages. The released benchmark comprises 600 screenshots organized into 95 state-coherent screenshot sets, with 665 application–category evaluation items spanning seven interaction categories. Rather than isolated page images, each set spans multiple routes and captures consistent cross-page state, providing the visual evidence required to reconstruct a complete runnable web application. To evaluate the resulting artifacts, we design an end-to-end evaluation chain that scores each along three dimensions: whether the application builds and renders (EXEC@1 / EXEC@3), how faithfully the rendered UI matches the reference (VFS), and whether the interactions implied by the screenshots are realized (IIS).

Of the three, IIS is the core measure and the hardest to design: static screenshots underdetermine behavior and admit multiple valid realizations, since the same visual state may correspond to different

correct implementations of interaction logic (e.g., a search box that can be triggered on Enter, button click, or real-time query). To address this challenge, we organize interactions commonly found in web applications into a taxonomy of seven categories and evaluate each with a rubric-based protocol along three complementary axes. Each inferred interaction is assessed against a category-specific rubric that checks whether the expected trigger, state transition, and resulting interface behavior are correctly realized. This taxonomy-driven design enables structured assessment of inferred behaviors without requiring a unique ground-truth execution trace, while providing fine-grained diagnostic signals beyond a single aggregate score.

We evaluate six frontier VLMs on UI2App. Even the strongest reaches an overall IIS of only $36.8$, indicating that inferring complete interaction behavior from static screenshots remains a frontier challenge. Moreover, visual fidelity does not imply interaction-inference capability: the leader on visual fidelity (VFS) places fourth on IIS, $4.5\times$ behind the IIS leader. The gap is widest on cross-route state: on $S3$-scope interactions (cross-route persistence), five of six models score at most $3.5$, and even the best reaches only $15.8$ out of $100$.

Our contributions are as follows:

- **Task formulation and benchmark.** We formulate interaction inference: inferring and realizing screenshot-implied interactions. We introduce UI2App, comprising $600$ screenshots in $95$ state-coherent sets for runnable, multi-route web applications. To our knowledge, UI2App is the first benchmark for interaction inference.
- **End-to-end evaluation protocol and IIS taxonomy.** We introduce a three-metric protocol spanning build executability (EXEC@1 / EXEC@3), visual fidelity (VFS), and interaction inference (IIS), with IIS built on the interaction taxonomy.
- **Extensive evaluation and analysis.** We benchmark six frontier VLMs and a Qwen2.5-VL scaling ladder, establishing the first baselines for image-only interaction inference and showing that visual fidelity does not imply interaction-inference capability, with cross-route state persistence a frontier-wide bottleneck.

## 2 Related Work

**Static Visual-to-Code.** Building on advances in screenshot and UI understanding (Lee et al., 2023; Baechler et al., 2024; Liu et al., 2024; Wu et al., 2024), vision-language models have made rapid progress on translating UI visuals into frontend code. Early benchmarks established the screenshot-to-code task with real-world webpages and visual similarity scoring (Si et al., 2025; Xiao et al., 2025b), then scaled the paradigm with synthetic and large-scale corpora (Yun et al., 2024; Gui et al., 2025). MRWeb extends generation to multi-page sites by pairing screenshots with a structured list of navigation links and resources (Wan et al., 2024). The underlying paradigm treats UI generation as a static rendering problem: tasks specify what an interface looks like, not how it behaves, and runtime interaction is left outside evaluation. In contrast, UI2App inherits the screenshot-based input but reframes the task around behavior, asking models to produce a runnable application whose interaction implementations are assessed alongside rendered appearance.

**Generation with Behavioral Guidance.** WebGen-Bench (Lu et al., 2025) and Artifacts-Bench (Zhang et al., 2025) specify functionality in text, while Vision2Web (He et al., 2026) pairs multi-screen prototypes with interaction descriptions. Interaction2Code (Xiao et al., 2025a), IWR-Bench (Chen et al., 2025), and WebCompass (Lei et al., 2026) provide pre-/post-interaction screenshots, interaction videos, and navigation annotations with state-change keyframes, respectively. These inputs provide behavioral guidance; UI2App instead requires interaction inference from static screenshots without behavioral annotations or transition demonstrations (Table 1).

**Framework-Based Application Generation.** Recent benchmarks extend evaluation beyond standalone HTML to framework-based code and applications, including DesignBench (Xiao et al., 2025b), WebGen-Bench (Lu et al., 2025), and Vision2Web (He et al., 2026). This setting better reflects component-based development, but requires consistent cross-file interfaces, resolvable dependencies, and successful compilation and bundling. UI2App uses a fixed React–TypeScript scaffold and evaluates executability alongside visual fidelity and screenshot-implied interactions.

Table 1: Comparison of benchmarks for web interface and application generation.

| Benchmark | Input modality | Dataset size | Output stack | Multi-page | No behavior hints | Interaction eval | Interaction eval size |
|---|---|---|---|---|---|---|---|
| **Visual reconstruction** | | | | | | | |
| Design2Code (Si et al., 2025) | Single image | 484 pages | HTML | ✗ | ✓ | ✗ | – |
| DesignBench–Gen (Xiao et al., 2025b) | Single image | 430 pages | HTML / Framework | ✗ | ✓ | ✗ | – |
| MRWeb (Wan et al., 2024) | Image + Text | 500 samples | HTML | ✓ | ✗ | ✗ | – |
| ComUIBench (Xiao et al., 2026) | Images | 150 pages / 40 sites | HTML | ✓ | ✓ | ✗ | – |
| **Interactive HTML generation** | | | | | | | |
| Interaction2Code (Xiao et al., 2025a) | Paired images | 127 pages | HTML | ✗ | ✗ | ✓ | 374 interactions |
| IWR-Bench (Chen et al., 2025) | Video | 113 tasks | HTML | ✗ | ✗ | ✓ | 1,001 actions |
| **Interactive application generation** | | | | | | | |
| WebGen-Bench (Lu et al., 2025) | Text | 101 tasks | Framework | ✓ | ✗ | ✓ | 647 test cases |
| Vision2Web–L2 (He et al., 2026) | Images + Text | 388 images / 66 apps | Framework | ✓ | ✗ | ✓ | 495 test cases |
| **UI2App (ours)** | **Images** | **600 images / 95 apps** | **Framework** | ✓ | ✓ | ✓ | **665 evaluation items** |

# 3 UI2App

## 3.1 Task Definition

A UI2App task supplies only a set of $M$ screenshots ($M = 3$–$14$, mean $6.3$), each capturing a distinct route or visual state of the target application. *No action trace or interaction description is provided.* Given this input, the model must produce the source code of a runnable application that reproduces the screenshots. All tasks and models share a fixed React + TypeScript scaffold, chosen as the most widely deployed front-end stack so that the benchmark reflects current production practice.

## 3.2 Dataset Construction

**Source pool and filtering.** We curate UI2App's reference screenshots from open-source GitHub projects. To obtain a broad initial pool, we use 24 archetype-aware GitHub search queries spanning 12 application categories, yielding 2,013 candidate repositories. We then apply a four-stage automated pipeline that checks permissive licensing, structural validity, buildability, and authentication-wall detection. This leaves 164 repositories for expert review.

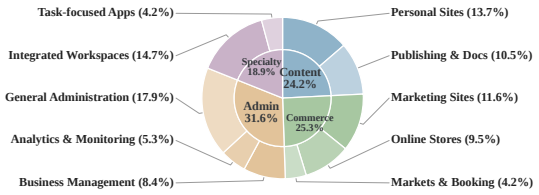



Figure 2: **UI2App dataset diversity.**

**Expert selection.** Experts select 95 of the 164 candidates using three criteria:

- **Page-level discriminability.** Pages should have non-trivial layouts, diverse content regions, atomic and composite UI components, and real text or images.
- **Application-level complexity.** Apps must offer distinct routes and screenshot-implied interactions across multiple IIS categories, and render stably at the capture resolution without overflow, unhydrated content, or dev-server artifacts.
- **Corpus-level diversity.** Reviewers exclude repeated visual themes, duplicated upstream starters, and near-identical templates to reduce layout and scaffold redundancy.

**Capture pipeline.** Each application is captured by a headless-browser script at $1440\times900$, with timing and content-validation rules producing post-hydration screenshots free of dev-server overlays (full pipeline in Appendix A). To assess generation from visual evidence, we exclude repository names, URLs, and reference source code from model inputs and evaluate visual fidelity and screenshot-implied functionality rather than source-code similarity.

**Stratification and statistics.** Applications are organised into four *application categories* by user-facing function: **Content**, **Commerce**, **Admin**, and **Specialty**. Subcategory listings appear in Table 8 (Appendix).

### 3.3 Evaluation Protocol

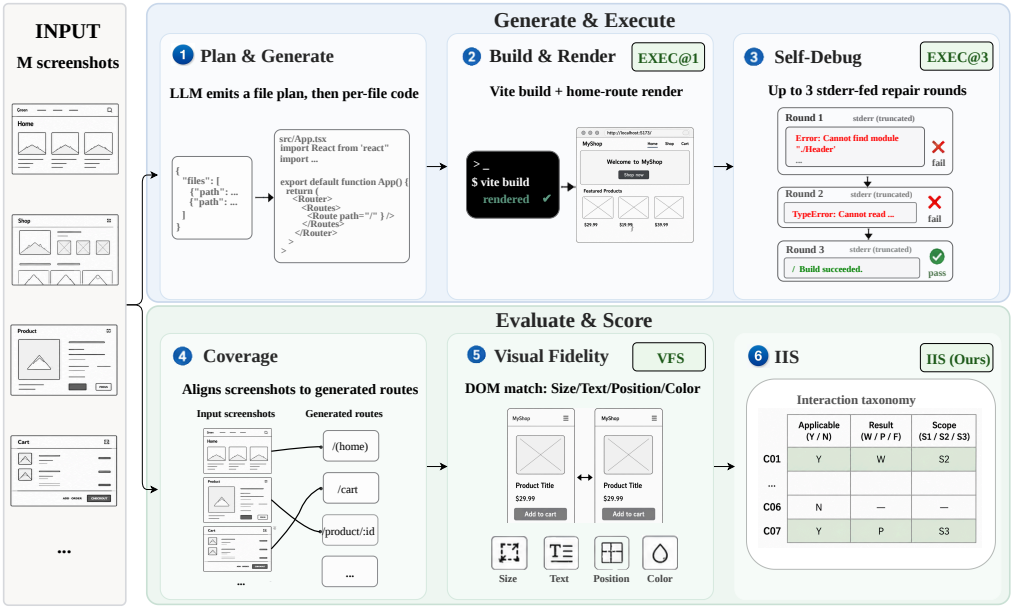



Figure 3: **The evaluation protocol of UI2App.**

The protocol decomposes evaluation into a sequence of stages. Figure 3 illustrates the full chain. (1) **Plan & Generate** splits generation into a file-list pass followed by per-file content. (2) **Build & Render** attempts a production build and a home-route render, producing the per-attempt EXEC pass/fail signal. (3) **Self-Debug** on EXEC failure feeds the build's error messages back to the same model for up to three repair attempts. (4) **Coverage** maps each input screenshot to its generated-app route and emits the page pairs consumed by VFS. (5) **Visual Fidelity** computes DOM-to-DOM similarity on the matched pairs, defining VFS. (6) **Automated interaction evaluation** uses GPT-6 Luna (medium) to label each interaction from generated source code and rendered screenshots along the interaction taxonomy.

### 3.4 Standard Metrics: EXEC, VFS

**EXEC@k (Executability).** EXEC measures whether the generated source code *actually runs*, the protocol's entry gate. An app passes when the production build completes without error and the home route renders meaningful content. EXEC@1 is the one-shot pass rate. EXEC@3, the pass rate after up to three rounds of error-feedback retry, separately measures iterate-fix capability.

**VFS (Visual Fidelity Score).** VFS scores *what is rendered where* via judge-free block-level matching: visible content blocks from each page are paired by optimal bipartite matching, and four sub-metrics (Size, Text, Position, Color) on the matched pairs are averaged unweighted. This design is robust to implementation differences in the underlying DOM. Algorithm and sub-metric formulas in Appendix F.

**Conventions.** *URL-direct loading*: Rendered screenshots for VFS and IIS are collected by loading each route directly via URL, isolating page implementation from routing. *Zero-imputation*: apps failing EXEC@3 contribute zero to VFS/IIS, keeping models with different EXEC@3 rates directly comparable.

### 3.5 Interaction Inference Score

**Definition.** IIS quantifies how well a model infers and realizes screenshot-implied interactions, jointly considering interaction coverage, implementation result, and state-logic complexity. Rather than comparing against a single reference behavior, IIS adopts a rubric-based evaluation in which each interaction is judged against the functional criteria. This implementation-agnostic formulation accommodates diverse yet semantically equivalent realizations of the same screenshot-implied interaction while enabling consistent evaluation across different applications.

**Interaction taxonomy and axes.** To make interaction inference measurable, we organize interactions commonly found in web applications into seven categories: toggle, expand/collapse, list operations, data CRUD, form validation, notification, and cross-route state (C01–C07, cataloged in Appendix B.1). Each category is evaluated along three axes: coverage, indicating whether the interaction category is implied by the screenshots; result, indicating whether the generated application realizes the interaction category (fully realized, partially realized, or failed); and scope ($S1$ UI-state, $S2$ data-state, $S3$ cross-route persistence), reflecting increasing state-management complexity in real front-end engineering. Separating these axes enables failures to be attributed to specific interaction types and aspects, rather than being collapsed into a single aggregate score.

**Scoring and aggregation.** For each application, we first construct a Reference Interaction Inventory (RII) from the input screenshots by labeling each interaction category with its coverage (whether the interaction is implied by the screenshots) and, when covered, its reference scope $s_i^{\text{ref}}$. The covered categories form the reference set $G_a$, ensuring that only interactions inferable from the screenshots are evaluated. For each generated application, every category is annotated along the same three axes: coverage $y_i \in \{0, 1\}$, implementation scope $s_i^{\text{gen}} \in \{1, 2, 3\}$, and result $r_i \in \{1, 0.5, 0\}$ (working, partial, or failed). Each scope level carries a linear weight, $w_{S1} = 1$, $w_{S2} = 2$, $w_{S3} = 3$. Combining the three axes yields the per-application quality score and the model-level IIS:

$$\mathrm{QS}_{a,m} = \frac{\sum_{i \in G_a} y_i \cdot \min\left(w_{s_i^{\text{gen}}},\ w_{s_i^{\text{ref}}}\right) r_i}{\sum_{i \in G_a} w_{s_i^{\text{ref}}}}, \qquad IIS_m = \frac{100}{N} \sum_{a=1}^{N} \mathrm{QS}_{a,m}, \tag{1}$$

with $N$ the number of applications. The denominator is the sum of reference scope weights, which makes IIS recall-oriented.

**Human IIS evaluation.** Two front-end engineering experts construct the RII from input screenshots alone. On a 45-application validation subset, two annotators independently assess each generated application's runtime behavior across all seven categories. Adjacent outcome disagreements are averaged; larger disagreements are adjudicated by a third annotator. Inter-annotator agreement is substantial (Krippendorff $\alpha$ in 0.72–0.84; Appendix B).

**Automated IIS evaluation.** To improve evaluation efficiency and support scalable, repeatable assessment, we implement Auto IIS with GPT-6 Luna (medium), which assigns category-level labels from generated source code and rendered screenshots without access to the generator identity. We validate these static assessments against human annotations obtained by executing interactions and observing runtime behavior on the 45-application validation subset. Coverage agreement is $86.4\%$; on items judged applicable by both evaluators, outcome agreement is $81.8\%$ and exact scope agreement is $91.1\%$. The resulting model-level IIS scores preserve the human-derived ranking of all six models, with a Pearson correlation of $0.991$ and a mean absolute error of $1.57$ IIS points (Appendix I).

## 4 Experiments

### 4.1 Experimental Setup

We evaluate six frontier VLMs: Claude Sonnet 4.6 (Anthropic, 2026), Kimi K2.5 (Kimi Team, 2026) (thinking mode enabled), GPT-5.4 (OpenAI, 2026), Gemini 3.1 Pro Preview (Google DeepMind, 2026), Qwen3.5-397B-A17B (Qwen Team, 2026), and GLM-4.6V (Z.ai, 2025). To make performance differences more directly attributable to the underlying models, we query each model through its vendor's official API under a unified generation protocol. The main table reports one generation

per application–model pair on all 95 applications, with IIS evaluated by GPT-6 Luna (medium). To validate automated scoring and assess generation variability, we use human annotations and three independent generations on a 45-application subset; evaluator agreement and repeated-generation results are detailed in Appendices I and J. We additionally evaluate four Qwen2.5-VL checkpoints on this subset to study within-family scaling (Section 4.5).

### 4.2 Main Results

**Overall ranking across the three metrics.** Table 2 shows the three-metric panorama. Claude Sonnet 4.6 leads EXEC@1 (95.8%) and EXEC@3 (100%). Gemini leads VFS (73.2) and Claude leads IIS (36.8), while GLM-4.6V is lowest on every metric. Visual and interaction quality yield different model orderings.

Table 2: **Per-category main results on UI2App.** All scores except EXEC (%) are on a 0–100 scale. Best per (category, column) shown with **sage cell + bold**; worst with blush cell.

| Category | Model | EXEC (%) | | | VFS components | | | | VFS | IIS |
|---|---|---|---|---|---|---|---|---|---|---|
| | | @1 | @3 | Δ | Size | Text | Pos. | Color | | |
| Content | Claude Sonnet 4.6 | **100.0** | **100.0** | 0.0 | 64.9 | 78.7 | 75.9 | 66.6 | 71.6 | **57.3** |
| | Gemini 3.1 Pro Preview | **100.0** | **100.0** | 0.0 | **71.1** | **82.2** | **79.8** | **69.8** | **75.7** | 13.7 |
| | Kimi K2.5 | 82.6 | 87.0 | +4.3 | 57.4 | 70.1 | 66.1 | 62.9 | 64.1 | 37.2 |
| | GPT-5.4 | 69.6 | 78.3 | +8.7 | 49.2 | 59.0 | 58.3 | 52.9 | 54.9 | 16.3 |
| | Qwen3.5-397B-A17B | 60.9 | 78.3 | +17.4 | 47.4 | 57.7 | 55.5 | 53.9 | 53.6 | 29.4 |
| | GLM-4.6V | 43.5 | 69.6 | +26.1 | 43.8 | 52.1 | 46.1 | 47.8 | 47.4 | 2.6 |
| Commerce | Claude Sonnet 4.6 | **95.8** | **100.0** | +4.2 | **69.1** | 75.0 | **74.6** | 68.8 | 71.9 | **36.7** |
| | Gemini 3.1 Pro Preview | **95.8** | **100.0** | +4.2 | 68.9 | **75.1** | 73.7 | **71.2** | **72.2** | 10.2 |
| | Kimi K2.5 | 87.5 | 87.5 | 0.0 | 61.8 | 67.3 | 65.4 | 65.0 | 64.8 | 12.5 |
| | GPT-5.4 | 83.3 | 91.7 | +8.3 | 59.8 | 67.5 | 66.5 | 65.7 | 64.9 | 7.8 |
| | Qwen3.5-397B-A17B | 83.3 | 87.5 | +4.2 | 58.5 | 63.7 | 62.7 | 60.6 | 61.4 | 15.0 |
| | GLM-4.6V | 20.8 | 62.5 | +41.7 | 39.9 | 40.0 | 40.6 | 38.3 | 39.7 | 10.3 |
| Admin | Claude Sonnet 4.6 | **90.0** | **100.0** | +10.0 | 68.4 | 77.1 | 78.1 | 62.1 | 71.4 | **21.3** |
| | Gemini 3.1 Pro Preview | 86.7 | 96.7 | +10.0 | **71.0** | **78.8** | **78.5** | **63.8** | **73.0** | 4.9 |
| | Kimi K2.5 | 60.0 | 66.7 | +6.7 | 43.9 | 49.1 | 49.9 | 39.5 | 45.6 | 13.9 |
| | GPT-5.4 | 56.7 | 73.3 | +16.7 | 47.7 | 54.5 | 54.7 | 45.5 | 50.6 | 1.1 |
| | Qwen3.5-397B-A17B | 70.0 | 76.7 | +6.7 | 54.6 | 59.5 | 59.0 | 49.6 | 55.7 | 6.9 |
| | GLM-4.6V | 20.0 | 46.7 | +26.7 | 25.9 | 29.4 | 29.2 | 25.7 | 27.6 | 3.5 |
| Specialty | Claude Sonnet 4.6 | **100.0** | **100.0** | 0.0 | **71.4** | 79.5 | 74.7 | 55.3 | 70.3 | **36.7** |
| | Gemini 3.1 Pro Preview | 72.2 | 94.4 | +22.2 | 65.6 | **81.8** | **80.1** | 59.1 | **71.7** | 3.7 |
| | Kimi K2.5 | 66.7 | 66.7 | 0.0 | 47.4 | 56.5 | 53.1 | 37.4 | 48.6 | 16.5 |
| | GPT-5.4 | 66.7 | 94.4 | +27.8 | 66.3 | 77.1 | 74.6 | **59.6** | 69.4 | 3.5 |
| | Qwen3.5-397B-A17B | 72.2 | 88.9 | +16.7 | 64.2 | 72.4 | 69.6 | 56.5 | 65.7 | 11.9 |
| | GLM-4.6V | 11.1 | 27.8 | +16.7 | 13.5 | 17.0 | 16.8 | 15.0 | 15.6 | 1.4 |
| **Overall** | Claude Sonnet 4.6 | **95.8** | **100.0** | +4.2 | 68.3 | 77.4 | 76.1 | 63.6 | 71.3 | **36.8** |
| | Gemini 3.1 Pro Preview | 89.5 | 97.9 | +8.4 | **69.5** | **79.3** | **77.9** | **66.3** | **73.2** | 8.1 |
| | Kimi K2.5 | 73.7 | 76.8 | +3.2 | 52.3 | 60.2 | 58.3 | 51.2 | 55.5 | 19.7 |
| | GPT-5.4 | 68.4 | 83.2 | +14.7 | 54.7 | 63.2 | 62.3 | 55.1 | 58.8 | 6.9 |
| | Qwen3.5-397B-A17B | 71.6 | 82.1 | +10.5 | 55.7 | 62.6 | 61.1 | 54.7 | 58.5 | 15.3 |
| | GLM-4.6V | 24.2 | 52.6 | +28.4 | 31.4 | 35.2 | 33.8 | 32.2 | 33.2 | 4.6 |

**EXEC self-debug heterogeneity.** Self-debug gains vary across models (Table 2): GLM-4.6V gains 28.4 points, GPT-5.4 gains 14.7, and Gemini and Qwen3.5 gain 8.4 and 10.5, respectively; Kimi gains 3.2 and Claude 4.2. GPT-5.4 still ranks 5/6 on IIS, indicating interaction deficiencies after build repair.

**VFS spread is largely build-driven.** Conditioning on EXEC@3-passing apps narrows the VFS range from 40.1 to 11.8 points and raises GLM-4.6V from 33.2 to 63.0 (Table 6). Appendix E breaks down the remaining visual differences.

**Consistency across generations.** On the 45-app subset, Gemini leads visual fidelity in all three generations, while Claude consistently leads IIS ($36.86 \pm 0.39$), exceeding Gemini by 27.91 points on average (95% hierarchical bootstrap CI: $[20.95, 35.06]$; Appendix J).

### 4.3 Interaction Inference Deep Dive

Through a fine-grained analysis of IIS across interaction categories, scope tiers, and application categories (Figure 4, Tables 3 and 9), we derive the four findings below.

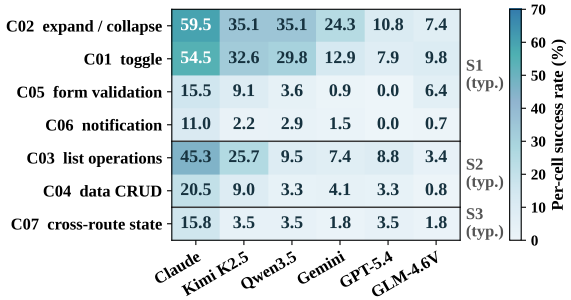



Figure 4: **IIS per-cell heatmap.** Per-(category, model) success rate ($\times 100$, capped, zero-imputation) over 7 categories $\times$ 6 baselines; rows grouped by typical scope (S1/S2/S3); columns ordered by overall IIS.

Table 3: $S$-restricted IIS (capped, zero-imputation, §3.5). Population per scope: apps with $\geq 1$ $S$-typical RII-positive item ($n = 95/80/57$). Best per column **bold**, worst underlined.

| Model | $S1$ | $S2$ | $S3$ |
|---|---|---|---|
| Claude Sonnet 4.6 | **39.9** | **37.8** | **15.8** |
| Kimi K2.5 | 24.1 | 19.7 | 3.5 |
| Qwen3.5-397B-A17B | 20.8 | 8.8 | 3.5 |
| Gemini 3.1 Pro Prev. | 10.8 | 8.4 | 1.8 |
| GPT-5.4 | 5.7 | 8.8 | 3.5 |
| GLM-4.6V | 6.6 | 2.8 | 1.8 |
| spread | 34.2 | 35.0 | 14.0 |

**Finding 1 — Visual fidelity does not imply interaction-inference capability.** The VFS leader and the IIS leader are different models: Gemini 3.1 Pro Preview ranks first on VFS (73.2) but fourth on IIS (8.1), while Claude Sonnet 4.6, second on VFS at 71.3, leads IIS at 36.8 (4.5$\times$ ahead of Gemini). Per-cell row-means (Figure 4) put Gemini at only 4.1, 1.5, and 1.8 on C04 (CRUD), C06 (notification), and C07 (cross-route state), showing that high VFS scores can coexist with missing core interaction logic.

**Finding 2 — The closed-vs-open quality hierarchy does not transfer to IIS.** Two of the three open-weight entries (Kimi 19.7, Qwen3.5 15.3) outscore the two closed-frontier entries below Claude (Gemini 8.1, GPT-5.4 6.9) on IIS, although both trail Gemini and GPT-5.4 on VFS. Visual-quality ordering therefore does not carry over to interaction inference.

**Finding 3 — Cross-route state is a shared gap among the evaluated models.** $S$-restricted IIS (Table 3) decreases as scope becomes harder: $S1 > S2 > S3$ holds for five of six models. The $S3$ end is a shared limitation: five models score only 1.8–3.5, and Claude reaches 15.8. Between-model spread contracts from 34.2 on $S1$ to 14.0 on $S3$. Cross-route state remains a challenging dimension of interaction inference.

**Finding 4 — Admin apps have the lowest average IIS.** Admin apps yield the lowest six-model mean IIS (8.6), compared with 26.1 on Content, 15.4 on Commerce, and 12.3 on Specialty (Appendix Table 9). Admin also has the narrowest between-model spread. Its low scores suggest a shared difficulty, although item composition and application complexity may contribute to the category gap.

### 4.4 Failure Analysis

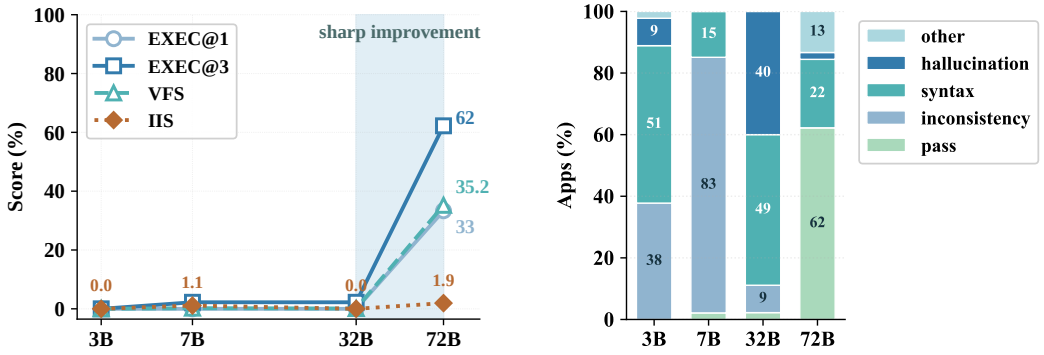
Across the six frontier models, distinct failure mechanisms emerge on the executability and interaction dimensions. Of the 101 EXEC@3 build/runtime failures, 38.6% violate one of four explicit plan-prompt constraints. Instruction compliance ranges from 100.0% to 71.6%. Hallucinated icon exports

and scaffold overwrites are the two largest failure categories. Cross-route state remains weak even among runnable applications. More detailed failure analysis is provided in Appendix G.

### 4.5 Within-Family Scaling (Qwen2.5-VL)

To assess how executability, visual fidelity, and interaction inference scale with model size, we evaluate Qwen2.5-VL models at 3B, 7B, 32B, and 72B on the 45-application subset. We use the vendor's official API and the same prompt templates for generation. We report EXEC@1, EXEC@3, VFS, and IIS, evaluating the latter with the main experiment's Luna protocol.

**Sharp improvement from 32B to 72B.** Across the $24\times$ size range, EXEC and VFS are essentially flat below 32B and jump sharply at 72B (Figure 5a): EXEC@3 from $\leq 2.2\%$ to $62.2\%$, VFS from $\leq 0.2$ to 35.2. Even at 72B, $37.8\%$ of generations still fail to build, indicating that within-family parameter scaling alone does not saturate UI2App's executability requirement. Interaction inference remains weak: IIS remains at 0.0, 1.1, 0.0, and 1.9 for 3B, 7B, 32B, and 72B, respectively.




(a) Scaling on four automatic metrics.

(b) Failure-mode composition.

Figure 5: **Qwen2.5-VL scaling on the 45-app subset (3B → 72B).** a: EXEC@1 (circles), EXEC@3 (squares), VFS (triangles, dashed), and IIS (diamonds, dotted); the shaded band marks the 32B → 72B improvement region. b: per-app build-failure category, stacked to 100%. Categories: *syntax*, *inconsistency*, *hallucination*, *other*, *pass*.

**Build-failure composition shifts with size, not just rate.** The dominant build-error category varies with model size (Figure 5b). The smaller checkpoints fail at basic source-code competence (3B on *syntax*, 7B on *inconsistency*), while the larger ones fail in more semantically loaded ways (32B on a mix of *syntax* and *hallucinated* dependencies, 72B at deeper build stages). Per-size counts and interpretation caveats are in Appendix G.2.

## 5 Conclusion

In this paper, we introduced UI2App, the first benchmark to measure *interaction inference*: recovering application behavior from image-only multi-page screenshots, with no textual or demonstration specification. Each artifact is scored along three dimensions (executability, visual fidelity, and IIS), with IIS grounded in the interaction taxonomy that tiers interactions by state-logic complexity. Across six frontier vision-language models, visual fidelity proves a poor proxy for interaction realization: the VFS leader trails the IIS leader by $4.5\times$. Cross-route state persistence is a bottleneck among the evaluated models, with five of six models scoring at most 3.5 and the strongest reaching only 15.8 on $S3$-scope interactions. UI2App provides a testbed for closing this $S3$ headroom. We hope it will drive vision-language models toward both stronger inference and more faithful realization of screenshot-implied interactions.

## AI USE STATEMENT

We used generative AI tools to assist with writing and polish the manuscript. GPT-6 Luna with medium reasoning effort is used as the automated IIS evaluator described in this paper. The authors take responsibility for the final content of this work.

## REFERENCES


Anthropic. Claude 4.6 sonnet, 2026. URL https://www.anthropic.com/news/claude-sonnet-4-6.

Gilles Baechler, Srinivas Sunkara, Maria Wang, Fedir Zubach, Hassan Mansoor, Vincent Etter, Victor Cărbune, Jason Lin, Jindong Chen, and Abhanshu Sharma. Screenai: a vision-language model for ui and infographics understanding. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 3058–3068, 2024.

Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems*, pp. 1–6, 2018.

Yang Chen, Minghao Liu, Yufan Shen, Yunwen Li, Tianyuan Huang, Xinyu Fang, Tianyu Zheng, Wenxuan Huang, Cheng Yang, Daocheng Fu, et al. Iwr-bench: Can lvlms reconstruct interactive webpage from a user interaction video? *arXiv preprint arXiv:2509.24709*, 2025.

Yuhong Dai, Yanlin Lai, Mitt Huang, Hangyu Guo, Dingming Li, Hongbo Peng, Haodong Li, Yingxiu Zhao, Haoran Lyu, Zheng Ge, et al. Webvr: Benchmarking multimodal llms for webpage recreation from videos via human-aligned visual rubrics. *arXiv preprint arXiv:2603.13391*, 2026.

Google DeepMind. Gemini 3.1 pro model card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan Wu, Xing Zhou, et al. Webcode2m: A real-world dataset for code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, pp. 1834–1845, 2025.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, YK Li, et al. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zehai He, Wenyi Hong, Zhen Yang, Ziyang Pan, Mingdao Liu, Xiaotao Gu, and Jie Tang. Vision2web: A hierarchical benchmark for visual website development with agent verification. *arXiv preprint arXiv:2603.26648*, 2026.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

Kimi Team. Kimi k2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset. *arXiv preprint arXiv:2403.09029*, 2024.

Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning*, pp. 18893–18912. PMLR, 2023.

Xinping Lei, Xinyu Che, Junqi Xiong, Chenchen Zhang, Yukai Huang, Chenyu Zhou, et al. WebCompass: Towards multimodal web coding evaluation for code language models. *arXiv preprint arXiv:2604.18224*, 2026. URL https://arxiv.org/abs/2604.18224.

Ryan Li, Yanzhe Zhang, and Diyi Yang. Sketch2code: Evaluating vision-language models for interactive web design prototyping. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3921–3955, 2025.

Junpeng Liu, Yifan Song, Bill Yuchen Lin, Wai Lam, Graham Neubig, Yuanzhi Li, and Xiang Yue. Visualwebbench: How far have multimodal llms evolved in web page understanding and grounding? In *First Conference on Language Modeling*, 2024.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch. *Advances in Neural Information Processing Systems*, 2025.

OpenAI. Gpt 5.4, 2026. URL https://openai.com/index/gpt-5-4-thinking-system-card/.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents, 2026. URL https://qwen.ai/blog?id=qwen3.5.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3956–3974, 2025.

Yuxuan Wan, Yi Dong, Jingyu Xiao, Yintong Huo, Wenxuan Wang, and Michael R Lyu. Mrweb: An exploration of generating multi-page resource-aware web code from ui designs. *arXiv preprint arXiv:2412.15310*, 2024.

Yifan Wu, Lutao Yan, Leixian Shen, Yunhai Wang, Nan Tang, and Yuyu Luo. ChartInsights: Evaluating multimodal large language models for low-level chart question answering. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 12174–12200, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.710. URL https://aclanthology.org/2024.findings-emnlp.710/.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu, Yuhang Wang, and Michael R Lyu. Interaction2code: Benchmarking mllm-based interactive webpage code generation from interactive prototyping. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 241–253. IEEE, 2025a.

Jingyu Xiao, Ming Wang, Man Ho Lam, Yuxuan Wan, Junliang Liu, Yintong Huo, and Michael R Lyu. Designbench: A comprehensive benchmark for mllm-based front-end code generation. *arXiv preprint arXiv:2506.06251*, 2025b.

Jingyu Xiao, Jiantong Qin, Shuoqi Li, Man Ho Lam, Yuxuan Wan, Jen-tse Huang, Yintong Huo, and Michael R. Lyu. ComUICoder: Component-based reusable ui code generation for complex websites via semantic segmentation and element-wise feedback. *arXiv preprint arXiv:2602.19276*, 2026. URL https://arxiv.org/abs/2602.19276.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Q Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, et al. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *Advances in neural information processing systems*, 37:112134–112157, 2024.

Z.ai. Glm-4.6v, 2025. URL https://huggingface.co/zai-org/GLM-4.6V.

Chenchen Zhang, Yuhang Li, Can Xu, Jiaheng Liu, Ao Liu, Changzhi Zhou, Ken Deng, Dengpeng Wu, Guanhua Huang, Kejiao Li, et al. Artifactsbench: Bridging the visual-interactive gap in llm code generation evaluation. *arXiv preprint arXiv:2507.04952*, 2025.

# A Dataset construction details

**Source-pool filter (§3.2).** The four-stage filter referenced in the main text applies the following per-stage criteria to each of the 2,013 candidate repositories:

1. **Permissive license.** The project's upstream `LICENSE` file (not package-manager metadata) must be MIT, Apache-2.0, BSD, or ISC.
2. **Structural validity.** The repository exposes a recognizable mainstream front-end framework layout with detectable component and route directories.
3. **Buildability.** Dependency installation and the production build both complete within a 180-second timeout on a clean environment.
4. **Authentication-wall detection.** Static heuristics over README/config files combined with HTTP-redirect probing of the served root.

Repositories surviving all four stages enter expert review. The cumulative pass rate is $164/2{,}013 \approx 8\%$.

**Screenshot capture pipeline.** Captures use Playwright with headless Chromium at viewport $1440\times900$ and `device_scale_factor`=1. The four-stage timing waterfall referenced in §3.2 is:

1. `domcontentloaded` plus a 3-second hydration settle;
2. injection of dev-overlay suppression CSS (Next.js HUD, Vite, webpack);
3. `wait_for_function` polling for asset readiness (6-second timeout);
4. animation completion via `document.getAnimations()` (3-second timeout), followed by a 13-selector loading-indicator detector with a 2-second recheck.

A per-route login-redirect detector complements the repo-level authentication-wall gate. Surviving screenshots are deduplicated by perceptual hashing on the content area ($32\times32$ average-hash, Hamming distance $< 6\%$).

# B IIS taxonomy, and inter-annotator agreement

## B.1 Category catalog

The seven interaction families spanned by IIS and their typical scopes are listed below.

| *Category* | *Description* | *Typical scope* |
|---|---|---|
| C01 toggle | show/hide, style switch, on/off | S1 |
| C02 expand/collapse | accordion, dropdown, drawer | S1 |
| C03 list operations | search, filter, sort, paginate | S2 |
| C04 data CRUD | create, read, update, delete | S2 |
| C05 form validation | required fields, format checks | S1 |
| C06 notification | toast, banner, confirmation feedback | S1 |
| C07 cross-route state | state persistence across navigation | S3 |

## B.2 State management and scope design

In production front-end engineering, interactive state is managed at increasing levels of architectural scope. The cheapest is *component-local state* (e.g., React `useState`/`useReducer`): state owned by a single component, bound to its mount lifecycle and not shared externally, such as whether a dropdown is open. When a piece of state is read or mutated by several sibling components, it must be *lifted to a shared owner* (a parent component, a React Context, or a store) so that updates stay consistent across the subtree; typical cases are a list with an active filter, or a selection shared between a toolbar and a table. The most demanding tier is *global state that must survive client-side route changes*: a single-page app re-mounts the route subtree on navigation, so any state that must persist (a cart, an auth session, a draft) cannot live in the page that gets unmounted and must instead live in a global store *whose lifecycle is independent of routing* (a Context at the app root, Redux/Zustand/Jotai,

or storage-backed). Implementation cost broadly increases along this hierarchy (a local handler, then cross-component state lifting, then a routing-independent global store), and the difficulty of *inferring* the requirement from the screenshots alone rises with it.

We turn this hierarchy into the three-level scope axis used by IIS, with an operational classification rule. An interaction is S1 (UI-state) when its effect is confined to the immediate rendered subtree (e.g., a toggle); S2 (data-state) when it performs a collection-level transformation (filter/sort/group/CRUD) that other components must observe; and S3 (cross-route persistence) when its effect must outlive a route change (e.g., a cart that stays non-empty after navigating to `/checkout`).

Because implementation cost broadly rises across S1<S2<S3, we weight each item by its scope with the linear ordinal weights $w_{S1}$=1, $w_{S2}$=2, $w_{S3}$=3 in Eq. 1, the simplest choice that respects this ordinal ranking without imposing an arbitrary curvature. The six-model IIS ordering is insensitive to this choice: recomputing model-level IIS under equal weights (1:1:1), the default (1:2:3), and an alternative (1:2:4) leaves the full ranking unchanged (Table 4; Kendall $\tau$=1.0 between every pair of weightings).

Table 4: **IIS is insensitive to the scope-weight choice.** Model-level IIS recomputed under three weightings of $(S1, S2, S3)$; the six-model ranking is identical, and Kendall $\tau$=1.0 between every pair of weightings. The 1:2:3 column reproduces the headline IIS of Table 2.

| Model | 1:1:1 | 1:2:3 (default) | 1:2:4 |
|---|---|---|---|
| Claude Sonnet 4.6 | 40.1 | 36.8 | 36.2 |
| Kimi K2.5 | 23.0 | 19.7 | 19.1 |
| Qwen3.5-397B-A17B | 18.5 | 15.3 | 15.0 |
| Gemini 3.1 Pro Preview | 9.7 | 8.1 | 7.9 |
| GPT-5.4 | 6.9 | 6.9 | 6.9 |
| GLM-4.6V | 5.5 | 4.6 | 4.5 |

### B.3 Outcome rubric and paired-merge interpolation

For each application, each interaction category is assessed through its constituent screenshot-implied interaction subtypes and tasks. Annotators evaluate these individual behaviors and assign a category-level outcome: *W* (working), all implied behaviors within the category are fully realized at runtime; *P* (partial), some but not all implied behaviors are realized, or at least one is only partially realized; and *F* (failed), none of the implied behaviors are realized. These map to the per-item score $r_i \in \{1, 0.5, 0\}$ used in Eq. 1. Under the paired design (§B, two annotators per cell), the cell's $r_i$ is set as follows: agreement uses the agreed value; a one-step disagreement (W/P or P/F) is merged by interpolation to $0.75$ or $0.25$, so that a partial implementation flagged by one annotator does not collapse to an extreme; a two-step disagreement (W/F) triggers arbitration by the third annotator, after which $r_i$ takes the arbitrated label.

### B.4 Inter-annotator agreement

The 45-app human validation subset contains 315 interaction-category items per model and 1,890 across six models. All generated applications in this validation subset were independently double-annotated, with a third annotator resolving disagreements. Inter-annotator agreement is assessed using paired annotations from the 45-application validation subset and summarized in Table 5. 95% bootstrap CIs are computed by resampling this subset with $B$=1,000, seed 42.

## C Provider-rejected projects

For GPT-5.4, one project was rejected by the provider before generation. It contributes zero to EXEC, VFS, and IIS in the 95-app main results. Excluding this project gives EXEC@3 $= 79/94 = 84.0\%$ and VFS $= 59.4$, versus $83.2\%$ and $58.8$ with zero-imputation. GPT-5.4 remains third on EXEC@3, between Gemini and Qwen. The 79 executable apps have conditional VFS $70.7$.

Table 5: Inter-annotator agreement on the three IIS annotation fields. `gen_appl` (the coverage axis) is binary so reports nominal $\alpha$ and Fleiss $\kappa$. `gen_result` (result) and `gen_scope` (scope) are ordinal so report $\alpha_{\text{ordinal}}$.

| Axis | Statistic | Value | 95% CI |
|---|---|---|---|
| `gen_appl` | $\alpha_{\text{nominal}}$ / Fleiss $\kappa$ | 0.77 / 0.73 | [0.71, 0.83] |
| `gen_result` | $\alpha_{\text{ordinal}}$ | 0.72 | [0.66, 0.78] |
| `gen_scope` | $\alpha_{\text{ordinal}}$ | 0.84 | [0.79, 0.88] |

## D CONDITIONAL METRICS: VFS$^\star$ AND IIS$^\star$

**Two denominator conventions.** UI2App reports two complementary scores for VFS and IIS. The headline scores in the Overall block of the main table average over the full curated set $N$=95 with zero-imputation: an application that fails to build after three self-debug rounds ($EXEC$@3=F) contributes 0 to the model's average, penalizing buildability and the underlying capability jointly. This is the most direct measure of end-to-end usability. The conditional variants VFS$^\star$ and IIS$^\star$ restrict the average to the $N^\star$ projects that pass executability ($EXEC$@3=P, i.e., the build and home-route render both succeed), isolating visual-language understanding and interaction-inference capability from buildability. We report both because the two factor different concerns into the same score:

- Headline (zero-imputation, $N$=95): is the rendered output of a complete model run usable on average?
- Conditional (VFS$^\star$/ IIS$^\star$, $N^\star \leq N$): given the model produced something at all, how visually faithful or interaction-faithful is it?

Table 6 reports VFS/VFS$^\star$ and IIS/IIS$^\star$ per model alongside their headline→conditional gaps. The headline→conditional gap is exactly the contribution of build failures to the headline score. The model with no build failures (Sonnet 4.6 at EXEC@3=100%) has $\Delta$=0 by construction; high-build-failure models open a large gap. GLM-4.6V's +29.9-point gap on VFS (33.2 → 63.0) is the most extreme: it has substantial visual-language ability ($N^\star$=50 projects average to 63.0), but only 52.6% of its outputs build at all, so its end-to-end VFS sits last. On IIS, the conditional score IIS$^\star$ similarly lifts GLM-4.6V from headline-last to mid-pack (4.6 → 8.7). The top three preserve their order (Claude 36.8 → 36.8, Kimi 19.7 → 25.6, Qwen3.5 15.3 → 18.7); the bottom three reshuffle because GLM's 50-app build-pass subset yields a higher per-app IIS than its full-set headline, moving it past Gemini (8.1 → 8.3) and GPT-5.4 (6.9 → 8.3) on the conditional view. The top-three preservation suggests that the IIS gaps among the strongest models in Table 2 reflect interaction-inference capability rather than executability alone.

Table 6: **Conditional metrics VFS$^\star$ and IIS$^\star$.** Both restrict the average to projects with EXEC@3=pass; $N^\star$ is the pass-conditioned project count. The headline→conditional gap separates executability from intrinsic capability. Best per column in **bold**, worst underlined; the largest absolute $\Delta$ (most build-constrained model) is bolded.

| Model | $N$ | EXEC@3 (%) | $N^\star$ | VFS | VFS$^\star$ | $\Delta_{VFS}$ | IIS | IIS$^\star$ | $\Delta_{IIS}$ |
|---|---|---|---|---|---|---|---|---|---|
| Claude Sonnet 4.6 | 95 | 100.0 | 95 | 71.3 | 71.3 | +0.0 | **36.8** | **36.8** | +0.0 |
| Gemini 3.1 Pro Preview | 95 | 97.9 | 93 | **73.2** | **74.8** | +1.6 | 8.1 | 8.3 | +0.2 |
| Kimi K2.5 | 95 | 76.8 | 73 | 55.5 | 72.2 | +16.7 | 19.7 | 25.6 | **+5.9** |
| GPT-5.4 | 95 | 83.2 | 79 | 58.8 | 70.7 | +11.9 | 6.9 | 8.3 | +1.4 |
| Qwen3.5-397B-A17B | 95 | 82.1 | 78 | 58.5 | 71.3 | +12.8 | 15.3 | 18.7 | +3.3 |
| GLM-4.6V | 95 | 52.6 | 50 | 33.2 | 63.0 | **+29.9** | 4.6 | 8.7 | +4.1 |

**Choice for the main table.** We headline VFS and IIS (zero-imputation) in Table 2 because end-to-end usability is the property the benchmark targets: a generation run that does not build is unusable to a real user, and its visual or interaction quality on hypothetical-builds-only is not actionable. VFS$^\star$ and IIS$^\star$ are reported as diagnostics that isolate visual-language and interaction-inference

capability from build success: they measure output faithfulness over only the projects a model builds, independent of how often it builds.

## E CONDITIONAL VISUAL SUB-METRIC BREAKDOWN

The main table reports the four zero-imputed VFS components, whose average reconstructs VFS up to stored-score rounding. In the Overall block, Gemini leads all four components, while GLM is lowest. Table 7 instead reports per-page means over matched pages in each model's executable subset; it diagnoses visual components without using the main table's app-level weighting.

Table 7: Conditional sub-metric breakdown. Per-page average over matched pages of EXEC@3-pass projects only; $n_{\text{pages}}$ varies per model and is reported alongside. Best per column in **bold** (ties shown in bold for both rows), worst underlined. GLM's scores draw from 235 pages (50 build-pass apps), versus Claude's 566 pages (95 apps); this diagnostic uses page weighting rather than the app weighting of VFS$^\star$.

| Model | $N^\star$ | $n_{\text{pages}}$ | Size | Text | Position | Color |
|---|---|---|---|---|---|---|
| Gemini 3.1 Pro Preview | 93 | 564 | **73.1** | **84.8** | **82.5** | 69.1 |
| Claude Sonnet 4.6 | 95 | 566 | 72.5 | 82.4 | 80.0 | 66.3 |
| Kimi K2.5 | 73 | 419 | 72.5 | 83.6 | 80.2 | 68.8 |
| GPT-5.4 | 79 | 477 | 72.5 | 84.7 | **82.5** | **72.6** |
| Qwen3.5-397B-A17B | 78 | 452 | 72.2 | 82.1 | 79.3 | 70.3 |
| GLM-4.6V | 50 | 235 | 67.3 | 77.9 | 74.5 | 70.6 |

The conditional view reveals two patterns invisible in the zero-imputation main table: (i) *Sub-skill gaps narrow once buildability is controlled for*: Position spread shrinks from 44.1 points (zero-impute) to 8.0 points (conditional), Color from 34.0 to 6.3. Visual sub-skill differences across the six models are several-fold smaller than overall VFS differences would suggest. (ii) *GLM-4.6V's signature is strong palette plus weak spatial composition*: it is last on Size and Position but second on Color (70.6, behind GPT-5.4's 72.6). Its relatively strong Color is conditional on a selected subset ($n_{\text{pages}}$=235 from 50 build-pass apps). These model-specific page populations do not identify performance on hypothetical repaired outputs.

## F VFS ALGORITHM: DOM-ALIGNMENT MATCHING AND CASCADE FALLBACK

**Overview.** VFS is a fully judge-free metric: no VLM, LLM, or pixel-classifier is invoked at any stage. Given a reference application $\mathcal{R}$ and a model-generated application $\mathcal{G}$ on the same input set $\mathcal{I}=\{(s_i, r_i)\}_{i=1}^{n^{\text{tot}}}$ (screenshot $s_i$ at reference route $r_i$), VFS is produced by the pipeline of Algorithm 1: (A) extract the route set of $\mathcal{G}$ from `src/App.tsx`; (B) boot the generated dev server and render each candidate route in headless Chromium under an auth/gate-dismissal protocol; (C) extract reference-side DOM text-blocks from $\mathcal{R}$, falling back to OCR on the input PNG when $\mathcal{R}$ cannot be reached at $r_i$ (auth-locked routes, missing dependencies); (D) match input screenshots to generated routes by *global Hungarian assignment* on the $M \times N$ matrix of reference-vs-generated DOM-block alignment scores, with a path-token Jaccard tie-breaker and an absolute floor; (E) score the aligned-block set of each matched pair on four sub-metrics (Size, Text, Position, Color); (F) aggregate to page-, application-, and model-level. A legacy cascade $T_1$–$T_5^b$ matcher is retained as a fallback for the rare case where $\mathcal{S}_{\text{ref}}$ fails to start (§F.1). All hyperparameters are held constant across all models and applications. Specific values are stated inline at each point of use.

**Auth and gate handling.** React applications in UI2App routinely guard non-public routes behind authentication or splash gates that, if not dismissed, cause every captured route to render the same login/intro view. We apply three source-derived mitigations on the generated dev server (no per-app human configuration), invoked at Algorithm 1 lines 2–4: (i) **`localStorage` auth seeding**: scan `src/**/*.[jt]s*` for storage call sites whose key matches the regex `/(auth|session|login|signin|loggedin|token|user)/i` and inject a permissive

auth payload via Playwright `add_init_script`; (ii) **in-memory auth form-fill**: when (i) finds no localStorage hooks but a `useState(false)` auth flag exists, navigate to `/login`, fill synthetic credentials, submit, and switch subsequent navigation to `pushState` so React state survives; (iii) **splash/intro gate dismissal**: AST-detect the `useState(false) + early return` pattern in `App.tsx`, regex-extract the dismiss-button label, and dismiss on every non-`/` route (the home is captured with the gate visible to match the reference). The same localStorage seeding is also applied to $\mathcal{S}_{\text{ref}}$; routes that seeding cannot bypass fall through to OCR.

**Render-validity probe.** For each rendered route we evaluate a four-signal probe on `#root`: `all` (descendant element count), `text` (`innerText` length), `visuals` (`svg/img/canvas/video` count), and `interactive` (`button/input/textarea/select/a[href]/[role="button"]` count). The route is accepted as rendered if `all`$\geq$3 *and* at least one of: `text`$\geq$20 chars, `visuals`$\geq$2, or `interactive`$\geq$2. The any-of disjunction (rather than a stricter conjunction such as "`text`$>10 \wedge$ `all`$\geq 5$") admits legitimate splash/CTA pages with shallow DOM but meaningful content (e.g., a `<div><button>Skip</button><p>...</p></div>` Intro component). Routes that fail the probe are recorded as `None` and become MISS candidates for the matcher.

**Route-alias deduplication.** After each successful screenshot we hash the page's `innerText`+child-count signature and consult a per-app `seen_dom_hashes` table. If the hash recurs we record the new route as an *alias* of the canonical route and reuse the canonical screenshot rather than writing a duplicate. This collapses redirects (`<Route path="/" element=<Navigate to="/foo">>`) and HashRouter mirror routes that would otherwise inflate $n^{\text{tot}}$ and double-count gen content.

**DOM-alignment matching (default path).** We extract DOM text-blocks for every rendered gen route (cached as $\mathcal{B}^{\text{gen}}_{\hat{r}}$) and, on the reference dev server, for every input route (cached as $\mathcal{B}^{\text{ref}}_{s_i}$). When the reference DOM at $r_i$ comes back empty (route auth-locked, dependency missing, build/dev-server failure on $\mathcal{R}$ at this route), we substitute OCR-derived blocks: Tesseract on the input PNG produces line-level boxes which are normalized to the same $\{$`text`, `bbox`, `fg_color`, `tag`$\}$ schema as the DOM extractor (Color is degraded for OCR-derived blocks since `fg_color` defaults to black). Let $M$ be the number of inputs and $N$ the number of rendered gen routes. We build the similarity matrix

$$\Sigma_{ij} \;=\; \text{compute_l3_page}(\mathcal{B}^{\text{ref}}_{s_i},\, \mathcal{B}^{\text{gen}}_{\hat{r}_j}) \,/\, 100, \qquad \Sigma'_{ij} \;=\; \min\big(1,\; \Sigma_{ij} + \beta \cdot \text{Jacc}(\tau(r_i), \tau(\hat{r}_j))\big),$$

where compute_l3_page is the four-sub-metric scorer (§*Sub-metrics* below), $\tau(\cdot)$ tokenizes a path on `[-_/]+`, and Jacc is set-Jaccard. The token bonus $\beta$=0.08 acts as a tie-breaker when DOM-alignment scores are within $\sim$0.05 of each other (e.g., gen `/leads` beats gen `/settings/team` for the input labeled `leads`, both candidates at $\Sigma \approx 0.55$), without overriding strong DOM evidence. We solve the assignment globally with the Hungarian algorithm (`scipy.optimize.linear_sum_assignment`) on cost $C = 1 - \Sigma'$ padded to the square size $\max(M, N)$ with cost 1, then drop pairs whose underlying $\Sigma_{ij}$ falls below the absolute floor $\theta_{\text{DOM}}$=0.20 (the input becomes MISS and contributes 0 to $n^{\text{matched}}$). Each gen route is used at most once across the assignment. This hard cap is equivalent to setting $K_{\max}$=1 (the same parameter that the cascade fallback uses, see §F.1) and prevents a single dynamic route (e.g., a `/post/:id` render) from absorbing three to five unrelated inputs and over-counting matched routes.

**DOM block extraction.** We navigate Playwright to the route under evaluation, wait until `domcontentloaded`, dismiss any detected gate (only on non-`/` routes; see auth-and-gate handling above), and idle for $\delta_{\text{ref}}$=4,000 ms (reference) or $\delta_{\text{gen}}$=3,000 ms (generated). Both renderings use viewport 1440$\times$900 to match the reference capture. We then run an in-page `TreeWalker` over text nodes and emit one block per visible parent element with: (i) lower-cased text trimmed to 200 characters; (ii) bounding box normalized to viewport so that $(x, y, w, h) \in [0, 1]^2 \times [0, 1]^2$; (iii) the foreground color from `getComputedStyle().color`; (iv) the parent's tag name. Non-rendering elements (`display:none`, `visibility:hidden`, opacity 0), boxes smaller than 2$\times$2 pixels, content more than three viewports below the fold (pixel-space $y_{\text{px}} > 3v_h$ where $v_h$=900 px is the viewport height), and text strings longer than 500 characters are filtered out before the bbox is normalized to $[0, 1]^2$. Adjacent same-row same-tag fragments are merged when within 0.02 normalized units, suppressing the spurious split of inline-styled text into multiple blocks.

**Block alignment within a pair.** Let $B^{\text{ref}}=\{b_i^{\text{ref}}\}_{i=1}^{n}$ and $B^{\text{gen}}=\{b_j^{\text{gen}}\}_{j=1}^{m}$. We build the cost matrix $C \in \mathbb{R}^{n\times m}$ with $C_{ij}=1-\rho(t_i^{\text{ref}}, t_j^{\text{gen}})$ where $\rho=$ `difflib.SequenceMatcher.ratio` is the Ratcliff–Obershelp similarity (longest common contiguous matching) $\rho(a,b)=2L/(|a|+|b|)$ with $L$ the matched-character count. We pad $C$ to the square size $\max(n,m)$ with cost 1, solve $(\sigma^*,\pi^*)=\arg\min\sum C_{\sigma\pi}$ via the Hungarian algorithm (`scipy.optimize.linear_sum_assignment`), and retain only assignments whose $\rho\geq\theta_{\text{text}}=0.3$. The retained assignment set $A$ is the basis for all four sub-metrics.

**Sub-metrics.** For a pair $p=(b_p^{\text{ref}}, b_p^{\text{gen}})$ let $a(b)$ denote area, $c(b)=(x+w/2,\ y+h/2)$ the center, $\text{fg}(b)$ the foreground color, and $\Delta E_{00}$ the CIEDE2000 color difference computed in CIE Lab via `skimage.color`. The four sub-metrics are

$$
\begin{aligned}
SIZE &= \min\left(1,\ \frac{\sum_{p\in A}\frac{1}{2}\big(a(b_p^{\text{ref}})+a(b_p^{\text{gen}})\big)}{\frac{1}{2}\Big(\sum_i a(b_i^{\text{ref}})+\sum_j a(b_j^{\text{gen}})\Big)}\right),\\
TEXT &= \frac{1}{|A|}\sum_{p\in A}\rho\big(t_p^{\text{ref}}, t_p^{\text{gen}}\big),\\
POSITION &= \frac{1}{|A|}\sum_{p\in A}\max\Big(0,\ 1-\max\big(|\Delta c_x|_p,\ |\Delta c_y|_p\big)\Big),\\
COLOR &= \frac{1}{|A|}\sum_{p\in A}\max\Big(0,\ 1-\Delta E_{00}\big(\text{fg}(b_p^{\text{ref}}), \text{fg}(b_p^{\text{gen}})\big)/100\Big).
\end{aligned}
$$

SIZE measures matched-block area coverage. We use Chebyshev (max-norm) displacement for Position rather than Euclidean because the four-metric average should treat horizontal and vertical drift symmetrically while remaining bounded in $[0,1]$ without an additional normalization constant. CIEDE2000 is divided by 100 since saturated complementary colors produce $\Delta E_{00}\approx 100$ in the Lab gamut. Each sub-metric defaults to 0 if $|A|=0$ (no aligned blocks).

**Aggregation.** The page-, application-, and model-level scores are

$$
\ell_p^{\text{page}} = 25\,(SIZE+TEXT+POSITION+COLOR), \quad L_{3,a}^{\text{matched}} = \frac{1}{|P_a|}\sum_{p\in P_a}\ell_p^{\text{page}},
$$

$$
VFS_{a,m} = L_{3,a,m}^{\text{matched}}\cdot\frac{n_{a,m}^{\text{matched}}}{n_a^{\text{tot}}}, \qquad VFS_m = \frac{1}{N_{\text{app}}}\sum_{a=1}^{N_{\text{app}}} VFS_{a,m}, \tag{2}
$$

with $N_{\text{app}}=95$ (the curated app count; the per-app local $N$ in Algorithm 1 denotes the number of rendered gen routes and is unrelated). The coverage factor $n^{\text{matched}}/n^{\text{tot}}$ penalizes models that achieve a high per-page score on a small subset of routes. For example, $1/12$ matched at $L_3^{\text{matched}}=31.7$ degrades to $VFS=2.6$ rather than retaining the per-page score. Consistent with the scoring policy of §3.3, applications with $EXEC@3=\text{F}$ are zero-imputed at the application level. The conditional variant VFS$^\star$ (Appendix D) drops the zero-imputation, averaging only over $EXEC@3=\text{P}$ applications.

### F.1 Cascade fallback ($T_1$–$T_5^b$, used only when $\mathcal{S}_{\text{REF}}$ fails to start)

When the reference dev server $\mathcal{S}_{\text{ref}}$ cannot be started for an application (missing `node_modules`, install timeout, port collision under heavy parallelism), the DOM-alignment matrix in Algorithm 1 is undefined and we fall back to a path-name + visual cascade. Each input is matched to at most one rendered gen route ($K_{\max}=1$, including dynamic routes) and a hit at tier $T_t$ skips $T_{t+1},\ldots,T_5^b$.

- $T_1$ **Exact route match.** The reference route $r_i$ (lower-cased, slashes/query stripped) equals the generated route's literal definition or its concretized URL.
- $T_2$ **Dynamic-route regex match.** Generated routes containing path parameters (`:slug`, `:id`) are converted to `[^/]+` regex. Catches `/product/1` ↔ `/product/:id`.

**Algorithm 1** Computation of VFS for a single (application, model) pair (DOM-alignment default; cascade fallback in §F.1).

**Require:** Input set $\mathcal{I}=\{(s_i, r_i)\}_{i=1}^{n^{\text{tot}}}$; reference source $\mathcal{R}$; generated source $\mathcal{G}$.
$\mathcal{T} \leftarrow$ ExtractRoutes($\mathcal{G}$/`src/App.tsx`) ▷ indent-aware nested-`<Route>` parser; concretize `:id`→`1`
$g \leftarrow$ DetectGates($\mathcal{G}$); $\sigma_g \leftarrow$ DetectLocalStorageSeeds($\mathcal{G}$); $\iota \leftarrow$ DetectInMemoryAuth($\mathcal{G}$) when $\sigma_g=\emptyset$
$\sigma_r \leftarrow$ DetectLocalStorageSeeds($\mathcal{R}$)
$\mathcal{S}_{\text{gen}} \leftarrow$ StartDevServer($\mathcal{G}$); apply seeds $\sigma_g$ via `add_init_script`
$H_{\text{gen}}$, $\mathcal{B}^{\text{gen}}$, aliases, hashes $\leftarrow \emptyset, \emptyset, \emptyset, \emptyset$
**if** $\iota$ **then** capture pre-auth Login at `__login__`; FormFillLogin($\mathcal{S}_{\text{gen}}$); switch to SPA `pushState`
**end if**
**for** $\hat{r} \in \text{sort}(\mathcal{T})$ ▷ / first, then by $|\hat{r}.\texttt{url}|$ **do**
  navigate $\mathcal{S}_{\text{gen}} \to \hat{r}.\texttt{url}$; if $\hat{r}.\texttt{url} \neq /$ **and** $g$.has_gate **then** TryDismiss($\mathcal{S}_{\text{gen}}$, $g$.selectors)
  probe $\leftarrow$ {`all`, `text`, `visuals`, `interactive`} on `#root`
  **if not** IsRendered(probe) **then** $H_{\text{gen}}[\hat{r}] \leftarrow$`None`; **continue** ▷ `all`$\geq 3 \wedge$ (`text`$\geq 20 \vee$ `visuals`$\geq 2 \vee$ `interactive`$\geq 2$)
  **end if**
  $h \leftarrow \text{md5}(\texttt{innerText} \mid \texttt{all})$; **if** $h \in$ hashes **then** aliases$[\hat{r}] \leftarrow \text{canon}(h)$; $H_{\text{gen}}[\hat{r}] \leftarrow H_{\text{gen}}[\text{canon}(h)]$; **continue**
  $H_{\text{gen}}[\hat{r}] \leftarrow$Screenshot($\mathcal{S}_{\text{gen}}$); $\mathcal{B}^{\text{gen}}_{\hat{r}} \leftarrow$ExtractBlocks($\mathcal{S}_{\text{gen}}$); hashes$[h] \leftarrow \hat{r}$
**end for**
**if** $\sigma_g \neq \emptyset$ **and** `__login__` $\notin H_{\text{gen}}$ **then** render / in a *fresh non-seeded* context, save as `__login__`
**end if**
StopDevServer($\mathcal{S}_{\text{gen}}$)
$\mathcal{S}_{\text{ref}} \leftarrow$StartDevServer($\mathcal{R}$); apply seeds $\sigma_r$; **if** fail **then goto** *cascade fallback* (§F.1)
$\mathcal{B}^{\text{ref}} \leftarrow \emptyset$
**for** $(s_i, r_i) \in \mathcal{I}$ **do**
  $b \leftarrow$ExtractBlocks($\mathcal{S}_{\text{ref}}, r_i$ with $g$-aware dismissal)
  **if** $b=\emptyset$ **then** $b \leftarrow$OCRBlocks($s_i$) ▷ Tesseract on input PNG; COLOR degraded to placeholder
  $\mathcal{B}^{\text{ref}}_{s_i} \leftarrow b$
**end for**
StopDevServer($\mathcal{S}_{\text{ref}}$)
$\Sigma \in \mathbb{R}^{M \times N}$ with $\Sigma_{ij} \leftarrow$ compute_l3_page($\mathcal{B}^{\text{ref}}_{s_i}, \mathcal{B}^{\text{gen}}_{\hat{r}_j}$)/100 (rendered routes only)
$\Sigma'_{ij} \leftarrow \min(1,\ \Sigma_{ij} + \beta \cdot \text{Jacc}(\tau(r_i), \tau(\hat{r}_j)))$ with $\beta$=0.08
$(\sigma^*, \pi^*) \leftarrow$Hungarian($1-\Sigma'$) on the $\max(M, N)$-square padded matrix (pad cost 1)
$P \leftarrow \{(s_i, \hat{r}_{j^*}) : \Sigma_{ij^*} \geq \theta_{\text{DOM}}=0.20\}$; $n^{\text{matched}} \leftarrow |P|$ ▷ each $\hat{r}$ used at most once
$\mathcal{L} \leftarrow [\,]$
**for** $(s_i, \hat{r}^*) \in P$ **do**
  $A \leftarrow$Hungarian($\mathcal{B}^{\text{ref}}_{s_i}, \mathcal{B}^{\text{gen}}_{\hat{r}^*}$; $\theta_{\text{text}}$=0.3) ▷ within-pair text-dissimilarity alignment
  $\ell^{\text{page}} \leftarrow 25\,(SIZE{+}TEXT{+}POSITION{+}COLOR)$; **push** $\ell^{\text{page}}$ to $\mathcal{L}$ ▷ Eq. (2)
**end for**
$L_3^{\text{matched}} \leftarrow \text{mean}(\mathcal{L})$; $VFS_{a,m} \leftarrow L_3^{\text{matched}} \cdot n^{\text{matched}}/n^{\text{tot}}$
**return** $VFS_{a,m}$

$T_3$ **Path-suffix match.** Largest common trailing-segment overlap $k \geq 1$. Admits `/dashboard` ↔ `/app/dashboard`.

$T_4$ **Synonym structural match.** Tokenize on `[-_/]` and expand by a fixed table of 32 synonym groups (e.g. {`home`, `landing`, `index`, `overview`}; {`contact`, `support`, `etc`, `misc`, `links`, `elsewhere`}; {`signin`, `signup`, `forgot`, `otp`, `verify`}); a hit requires non-empty intersection with at least one cross-direction unexpanded-vs-expanded overlap.

$T_5^a$ **Visual-first phash Hungarian.** Build the $M \times N$ `phash`-similarity matrix on $16 \times 16$ perceptual hashes of input PNG vs gen screenshot, solve Hungarian on $1 - \text{sim}$, and accept pairs with $\text{sim} \geq \theta_{\text{phash}}=0.45$.

$T_5^b$ **Visual fallback (legacy SSIM+hist).** For inputs unmatched after $T_1$–$T_5^a$, compute $0.7\,\text{SSIM} + 0.3\,\text{HistCorr}$ on $256 \times 160$ resized RGB; accept the best candidate *only if* it satisfies both the relative margin $\text{best} \geq 1.2\bar{\mu}$ and the absolute floor $\theta_{\text{vis}}^{\text{abs}}=0.55$. The absolute floor blocks "least bad" assignments from a uniformly poor candidate set. For instance, it rejects a `zen-keyboard` screenshot pairing to `/player` at SSIM+hist$\approx$0.45.

**Implementation notes and known limitations.** The DOM-alignment default is robust to *route renaming* (gen `/` that actually serves Blog content correctly pairs with the reference's `/blog` input via DOM evidence rather than path string) and rejects *path-name false positives* (gen `/` that serves Blog cannot pair with a reference `/` that serves a music landing). The token-Jaccard bonus $\beta$=0.08 is small enough that DOM evidence dominates whenever it is decisive, and large enough to break ties in the noisy 0.5–0.65 similarity band typical of dark-theme/icon-heavy apps. The OCR fallback for $\mathcal{B}^{\text{ref}}$ degrades Color (no real foreground color information) but preserves Size, Text, and Position. This is the maximal signal recoverable for routes the reference dev server cannot reach. Because input screenshots in UI2App are URL-reachable static routes (§3.2), the pipeline does not replay click sequences before `ExtractBlocks`. Interaction-state fidelity is captured by IIS (§3.5) rather than VFS. We report VFS as observed under this protocol. An interaction-aware variant is left to future work.

Table 8: **Application-category composition by subcategory.** The four application categories resolve into the subcategories listed below, spanning the realized dataset diversity. Subcategories are descriptive and not used as analytical strata.

| Category | Share | Subcategories |
|---|---|---|
| Content | 24.2% | Personal Sites, Publishing & Docs |
| Commerce | 25.3% | Marketing Sites, Online Stores, Markets & Booking |
| Admin | 31.6% | Business Management, Analytics & Monitoring, General Administration |
| Specialty | 18.9% | Integrated Workspaces, Task-focused Apps |

Table 9: IIS by application category. Mean and spread summarize the six model means.

| Category | Apps | Mean | Spread | Claude | Kimi | Qwen | Gemini | GPT | GLM |
|---|---|---|---|---|---|---|---|---|---|
| Content | 23 | 26.1 | 54.7 | 57.3 | 37.2 | 29.4 | 13.7 | 16.3 | 2.6 |
| Commerce | 24 | 15.4 | 28.9 | 36.7 | 12.5 | 15.0 | 10.2 | 7.8 | 10.3 |
| Admin | 30 | 8.6 | 20.3 | 21.3 | 13.9 | 6.9 | 4.9 | 1.1 | 3.5 |
| Specialty | 18 | 12.3 | 35.3 | 36.7 | 16.5 | 11.9 | 3.7 | 3.5 | 1.4 |

# G Failure Taxonomy Details

This appendix expands the failure analysis of §4.4 into a breakdown across the benchmark's three scoring axes. We first decompose the 101 EXEC@3 build/runtime failures into a nine-mode taxonomy (Table 10) with per-model incidence, prompt-clause attribution, and representative code anchors, then trace how the dominant mode shifts along the Qwen2.5-VL size ladder. We next specify the fixed self-debug feedback protocol under which EXEC@3 is measured, and the recurring patterns that reduce VFS and IIS. A closing set of case studies illustrates the latent-affordance inference that IIS targets. Counts are over the 570 nominal (app, model) generation pairs spanning 95 apps and 6 baselines; one additional (GPT-5.4, app) pair rejected by the vendor's content-safety filter is treated separately (Appendix C) and is excluded from these 101 failures.

## G.1 Generation-level failures

The 101 EXEC@3 failures decompose into nine fine-grained modes (Table 10), which collapse into five high-level causes: hallucinated imports (C1, C6); scaffold-file overwrites (C2, C8); undeclared / wrong-name source dependencies (C3); broken import paths (C4, C5); and parse-level or runtime errors (C7, C9).

**C1: hallucinated `lucide-react` icon export (34.7% of failures).** The model imports a symbol not exported by the scaffold-pinned `lucide-react@^0.460.0`: typically a brand icon (`Tiktok`, `Spotify`, `Weibo`, `Alipay`, etc.), a future-renamed name (`Funnel`, the post-`0.460` canonical name introduced when the icon previously exported as `Filter` was renamed), or a foreign-library identifier (`DashboardOutlined` from Ant Design). An-

Table 10: **Build-failure modes across** 101 **EXEC@3 failures.** The *Share* column gives the mode's percentage of all 101 failures; per-model columns give the mode's incidence as a percentage of that model's 95 apps. "IFV" marks instruction-following violations against the four explicit plan-step prompt constraints (Appendix G.1). Models: *C*=Claude Sonnet 4.6, *Ge*=Gemini 3.1 Pro Preview, *Gp*=GPT-5.4, *K*=Kimi K2.5, *Q*=Qwen3.5-397B-A17B, *GL*=GLM-4.6V. One provider rejection is excluded from attribution and scored zero in the main results. Percentages are rounded to the nearest integer, so the *Share* column need not sum to exactly 100% and a model's per-mode entries need not sum exactly to its *Failure rate*.

| ID | Failure mode | Share | IFV | C | Ge | Gp | K | Q | GL |
|---|---|---|---|---|---|---|---|---|---|
| C1 | Hallucinated icon export (`lucide-react`) | 35% | | 0 | 1% | 11% | 11% | 11% | 4% |
| C2 | Scaffold `package.json` overwritten | 25% | ✓ | 0 | 1% | 0 | 1% | 4% | 20% |
| C3 | Pkg used but not in `extra_dependencies` | 11% | ✓ | 0 | 0 | 0 | 3% | 3% | 5% |
| C4 | `@/*` path alias used | 2% | ✓ | 0 | 0 | 0 | 0 | 0 | 2% |
| C5 | Wrong relative path | 5% | | 0 | 0 | 0 | 0 | 0 | 5% |
| C6 | Cross-file import/export mismatch | 6% | | 0 | 0 | 0 | 1% | 0 | 5% |
| C7 | JSX / TypeScript parse error | 10% | | 0 | 0 | 1% | 6% | 0 | 3% |
| C8 | Scaffold `src/index.css` overwritten | 1% | ✓ | 0 | 0 | 0 | 0 | 0 | 1% |
| C9 | Runtime API misuse | 6% | | 0 | 0 | 4% | 1% | 0 | 1% |
| *Failure rate* (failures / 95 apps) | | | — | **0** | 2% | 16% | 23% | 18% | 47% |
| *ICS* $= 1 - n_{\text{IFV}}/n_{\text{apps}}$ | | | — | **100%** | 99% | **100%** | 96% | 93% | 72% |

chor: GPT-5.4 on `55_shadcn-nextjs-dashboard` writes `import { Funnel } from "lucide-react"`, but in `0.460.0` the icon is still exported as `Filter`.

**C2: scaffold `package.json` overwritten** (24.8% **of failures).** The model emits a `package.json` in violation of the "Do NOT include scaffold files" clause, pinning unpublished versions or inventing package names (e.g., `mantine` as a top-level package; registry name is `@mantine/core`). `pnpm install` aborts with `ERR_PNPM_NO_MATCHING_VERSION` / `ERR_PNPM_NO_VERSIONS`. Anchor: GLM-4.6V on `60_mantine-dashboard`.

**C3: source uses pkg not in `extra_dependencies`** (10.9%)**.** The plan-step prompt requires a structured `extra_dependencies` field for non-scaffold packages, but the source imports a package neither pinned in the scaffold nor declared. Declaring non-scaffold dependencies is the model's responsibility under the plan-step contract, and the Self-Debug stage does not backfill the missing declaration by scanning generated source. Anchor: GLM-4.6V on `54_slash-admin` declares `react-calendar` but imports `react-big-calendar`.

**C4: `@/*` path alias used** (2.0%)**.** The model writes shadcn-style imports (`import X from "@/components/X"`) although the prompt requires relative imports and the scaffold's `tsconfig.json` declares no such alias.

**C6: cross-file export name mismatch** (5.9%)**.** The importer references an identifier whose source declaration uses a different name (default vs. named, casing mismatch, or omitted altogether). Anchor: GLM-4.6V on `38_dexkit-nft-marketplace` uses `<CollectionCard>` at `Home.tsx:31` without a matching `import`. `vite build` passes (esbuild does not type-check undefined identifiers) but the home route throws `ReferenceError` on first render, failing EXEC's home-route render gate (§3.4).

**Long-tail modes C5, C7, C8, C9** (21.8% **combined).** C5 (wrong relative path): file-root-relative specifier instead of `../`. C7 (JSX/TS parse error): unbalanced closing tag, stray `>`. C8 (scaffold `src/index.css` overwritten): `@layer base` without `@tailwind base`. C9 (runtime API misuse): GPT-5.4 on `18_myPortfolio` destructures a non-existent `location` field from the `<NavLink className={...}>` render-prop, which in `react-router-dom@6` provides `{ isActive, isPending, isTransitioning }`.

**Per-model signature and prompt-clause attribution.** Per-model totals (Table 10) reveal distinct signatures: Claude produces zero failures, while Gemini has two; GPT-5.4 (15.8% of its 95 apps) is dominated by C1 (66.7% of its failures, all factuality); GLM-4.6V (47.4% of its apps) is dominated by scaffold-respect violations (C2+C3+C4+C8 60.0%). The plan-step prompt (Appendix H) states four mechanically-checkable constraints: (i) source files under `src/`; (ii) relative imports; (iii) every non-scaffold import declared in `extra_dependencies`; (iv) no regeneration of scaffold files. Of the 101 failures, 39 (38.6%) are instruction-following violations (IFV column of Table 10): clause (iv) accounts for 66.7% of IFVs via C2 + C8, clause (iii) for 28.2% (C3), clause (ii) for 5.1% (C4), and clause (i) has zero violations. The remaining 62 failures (C1, C5, C6, C7, C9) are coding-quality or factuality errors not pre-empted by the prompt. The Instruction-Compliance Score (ICS) row shows the resulting model-level split.

### G.2 Per-size failure-mode shifts on the Qwen2.5-VL ladder

On the 45-app scaling subset, Figure 5b composes the build-failure category at each size, supporting the dominant-shift narrative in §4.5: the dominant category shifts from *syntax* (3B) to *inconsistency* (7B) to a mixed *syntax* + *hallucination* regime (32B). The post-transition 72B residual is dominated by *syntax* at deeper build stages plus the figure's *other* bucket, here *runtime* errors (e.g., importing TypeScript-only types as values). These four buckets correspond to the C1–C9 taxonomy of §G.1 as: *syntax*$\supseteq$C7, *inconsistency*$\supseteq$\{C2,C3,C4,C5,C8\}, *hallucination*$\supseteq$\{C1,C6\}, *other*$\supseteq$C9. We caution that the breakdown at $\leq$ 32B is over a near-fully-failing population ($\geq 44/45$ failures per size), so the precise within-size category proportions are noisy; the identity of the dominant (mode) category, taken over 44–45 failures, is well determined and is what the cross-size dominant-shift claim relies on.

### G.3 Self-debug feedback protocol

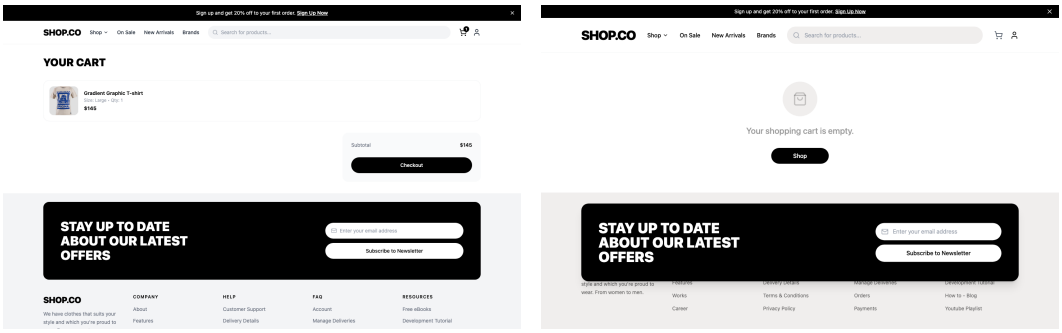
EXEC@3 is measured under a Self-Debug stage configured identically for every model: each failing app is retried for up to three rounds with the same model, and each round forwards the toolchain's own diagnostics back to the model for repair. EXEC@3 therefore reflects iterate-and-fix ability under this fixed feedback protocol rather than the model in isolation; how reliably a model localizes and repairs the root cause from standard build output is part of what it measures, with weaker self-debuggers returning `NO_FILE`.

### G.4 Visual-fidelity failures

Among $EXEC@3$-pass apps, the sub-skill view is best read off the conditional breakdown Table 7 (the zero-imputation columns of Table 2 are dominated by build-success differences, not sub-skill). Conditional spreads are tight: Color 66.3–72.6 (6.3), Position 74.5–82.5 (8.0), Size 67.3–73.1 (5.8), Text 77.9–84.8 (6.9). The GLM case illustrates reference-subset rendering rather than horizontal grid collapse: the generated DOM contains a strict subset of the reference's blocks, so missing reference blocks reduce visual coverage (see the GLM-4.6V case study in Fig. 13). Color drops by mis-derived theme palettes when screenshots use a non-default brand color. GLM-4.6V is the salient outlier: last on Size and Position but second on Color among build-pass apps. Palette extraction on the matched subset is strong, but the generated DOM covers only part of the reference.

### G.5 Interaction-inference failures

Figure 6 anchors a representative IIS failure on a shared input. Two models receive the same five e-commerce screenshots of `48_shopco-ecommerce`; an annotator then performs the same Add-to-Cart action on the product page and navigates to `/cart`. Claude Sonnet 4.6 (left) threads the action through a `CartContext` and the cart route renders the added product (`result=W`). Gemini 3.1 Pro Preview (right) emits a visually-faithful product page with an `Add to Cart` button whose handler does not propagate to any cross-route store, so the cart route still renders its empty-state placeholder (`result=F`). The cart route's static screenshot at protocol time is therefore indistinguishable from the reference's empty-cart capture.




(a) Claude Sonnet 4.6, `/cart`: product persists across the route boundary.

(b) Gemini 3.1 Pro Preview, `/cart`: empty-state placeholder; handler did not reach a cross-route store.

Figure 6: **Interaction-inference failure on `48_shopco-ecommerce`.** Identical input screenshots and identical click sequence; the divergence is the *Broken shared state* pattern. VFS on the static cart-route capture cannot distinguish the two outputs (both render an empty cart at protocol time), whereas IIS flags the missing S3 affordance.

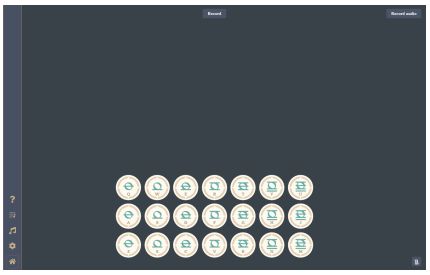
### G.6 Case studies: latent-affordance inference

The eight case studies below *illustrate* four claims from the main text: differences in interaction implementation, domain-level behavioral inference, build-time failure modes, and VFS sub-metric divergence; they are chosen to span these four claim types, and the aggregate evidence for each lives in the corresponding section and table (§4.3, Tables 10 and 7).

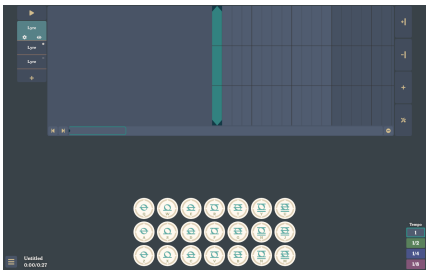
**Audio synthesis (`31_genshin-music`).** The `31_genshin-music` task supplies 9 screenshots of a stylized in-browser music synthesizer (Fig. 7): a hexagonal note pad, a key-binding overlay, a composer timeline, and a full-screen "Zen mode" player. None of the screenshots state explicitly that the application produces sound. The audio capability must be inferred from layout cues alone: musical symbols on each pad, MIDI-style key labels (`Q–U` / `A–J` / `Z–M`), an instrument-picker dropdown, and a decay-envelope visualization behind each note.

Of the six baselines, four reproduce the interface as a runnable static visual replica: pads animate on hover, but `onMouseDown` handlers are absent or no-ops and the emitted `package.json` declares no audio dependencies; GLM-4.6V fails to build. Only Kimi K2.5 instantiates a complete Web-Audio signal chain (`AudioContext` → triangle-wave `OscillatorNode` → `GainNode` with a 500 ms exponential-decay envelope) and wires keyboard events together with pointer events on every pad through a MIDI-to-frequency converter $f = 440 \cdot 2^{(n-69)/12}$. The artifact is a functionally playable synthesizer rather than a screenshot.

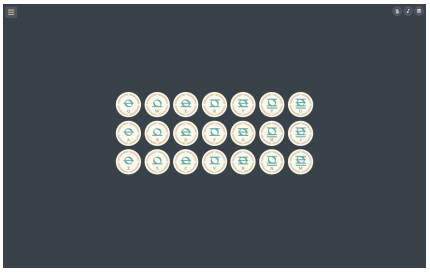
The contrast illustrates the regime IIS (§3.5) is designed to expose. The rendered interfaces are visually comparable and VFS cannot separate them, yet only one artifact is functionally playable. Crucially, no input screenshot can be flagged as "wrong" for the four UI-only models, since their renderings reproduce the visual targets' structure: grid, labels, and layout. The gap is one of behavior inferred from cues that the visual modality under-specifies, precisely the regime UI2App is built to expose for multi-modal code-generation agents beyond surface mimicry. Figure 8 contrasts the source for one pad cell across the two regimes (the UI-only listing is from Gemini 3.1 Pro Preview, representative of the four visually-faithful-but-silent baselines).



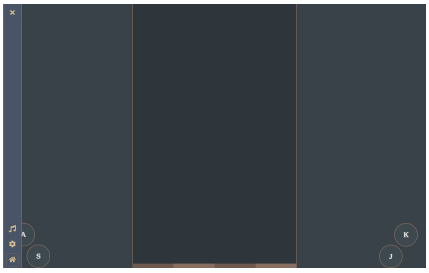
/



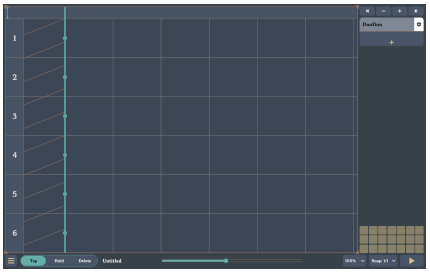
/composer



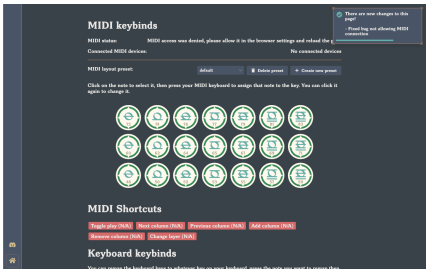
/zen-keyboard



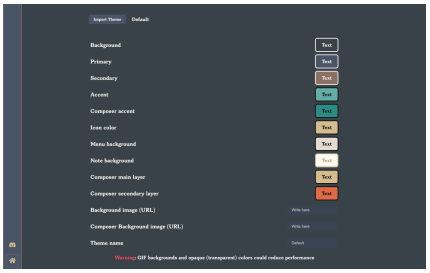
/vsrg-player



/vsrg-composer



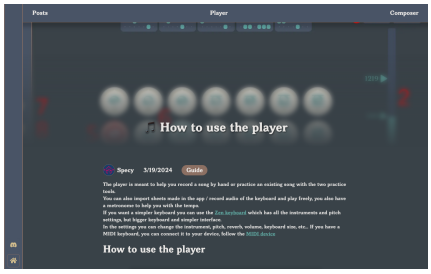

/keybinds



/theme




/blog




/blog/posts/...

Figure 7: **Input set for `31_genshin-music` (9 screenshots, one per route).** The audio capability must be inferred from these static cues alone: note pads with musical glyphs and MIDI key labels (Q–U/A–J/Z–M) on `/zen-keyboard` and `/keybinds`, a composer timeline (`/composer`, `/vsrg-composer`), and a full-screen player (`/vsrg-player`).

**src/components/Keyboard.tsx**

```
function KeyButton({ label, ... }) {
  return (
    <button className="w-[72px] h-[72px]
                       rounded-full bg-[#
    ebdcb8]
                       active:scale-95 ...
    ">
      <div className="rounded-full
                      bg-[#f8f5ea] border
    -2
                      flex flex-col ...">
        <svg width="24" height="24" ...>
          <circle cx="12" cy="12" r="9"
    />
          <line x1="4" y1="12"
                x2="20" y2="12" />
        </svg>
        <span className="...">
          {label}
        </span>
      </div>
    </button>
  );
  // no onClick / onMouseDown
}
```

Listing 1: (b) Gemini 3.1 Pro Preview: `src/components/Keyboard.tsx` (extract). Pad is a styled `<button>` with no `onClick`/`onMouseDown`; the SVG and key label are decorative.

**src/hooks/useAudio.ts**

```
const playNote = useCallback(
  (midiNote: number) => {
    initAudio();
    const ctx = audioContextRef.current;
    if (!ctx) return;
    stopNote(midiNote);
    const osc = ctx.createOscillator();
    const gain = ctx.createGain();
    const freq = 440 * Math.pow(
      2, (midiNote - 69) / 12);
    osc.type = 'triangle';
    osc.frequency.setValueAtTime(
      freq, ctx.currentTime);
    gain.gain.setValueAtTime(
      0.3, ctx.currentTime);
    gain.gain.
     exponentialRampToValueAtTime(
      0.01, ctx.currentTime + 0.5);
    osc.connect(gain);
    gain.connect(ctx.destination);
    osc.start();
    oscillatorsRef.current.set(midiNote,
     osc);
    setTimeout(() => stopNote(midiNote),
     500);
  }, [initAudio]);
```

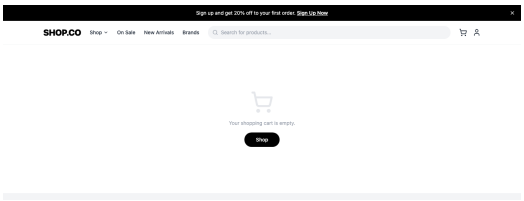
Listing 2: (c) Kimi K2.5: `src/hooks/useAudio.ts` (extract). `playNote` builds the Web-Audio chain (`OscillatorNode` → `GainNode`, 500 ms decay) with a MIDI-to-frequency converter.

Figure 8: **Latent-affordance inference on `31_genshin-music`.** Four of six baselines emit (b)-style code: visually-faithful pads with no audio. Only Kimi K2.5 instantiates the (c)-style Web-Audio signal chain, `AudioContext` → `OscillatorNode` → `GainNode` with a 500 ms exponential decay, and a MIDI-to-frequency converter $f = 440 \cdot 2^{(n-69)/12}$. VFS cannot read this gap from the rendered DOM; the missing audio runtime is a latent affordance that surfaces only under behavioral inspection, the inference regime IIS targets (§3.5).

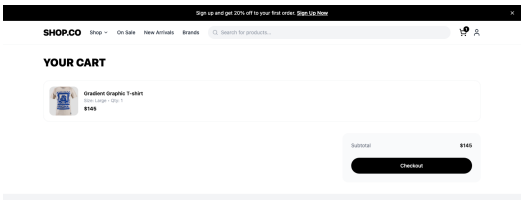
**State-machine inference (`01_fashion-ecommerce`, `48_shopco-ecommerce`).** Two e-commerce tasks, `01_fashion-ecommerce` (6 screenshots: home, shop, product detail, cart, checkout, search) and `48_shopco-ecommerce` (5 screenshots: home, shop, cart, and two product details), require an analogous inference along an orthogonal direction. Rather than an audio runtime, the model must infer the underlying *state machine* that backs the interface: a global cart that survives route changes (S3), variant selection on the product page that propagates into add-to-cart payloads (S2), search and sort on the shop (S2), and controlled inputs on a checkout form. As with audio, the screenshots never declare these affordances explicitly: a "$0.00" subtotal hints at a reactive total, a size grid at selectable variants, a search bar at client-side filtering.

Across interaction items spanning the two projects, Claude Sonnet 4.6 produces lots of interaction components that work end-to-end: a `CartContext` de-duplicating items by the composite key (*id*, *size*, *color*), auto-clearing entries when quantity drops to zero, a controlled `CheckoutPage` form, and a search page with submit-driven filtering on product name and category. The next-best on these two tasks, GPT-5.4, supplies a partially wired `ShopContext` on `48_shopco-ecommerce`, but its `addToCart` mutator is never read by the cart page, which renders a static empty-state placeholder, dead code in the literal sense. The remaining four baselines (Gemini 3.1 Pro Preview,

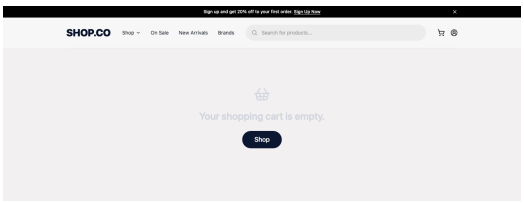
GLM-4.6V, Qwen3.5-397B-A17B, Kimi K2.5) systematically emit visually faithful but inert components: add-to-cart buttons without `onClick`, cart pages reduced to "*Intentionally left blank*" placeholders, and no cross-component state container at all. Figure 9 contrasts the `/cart` render across the three regimes.



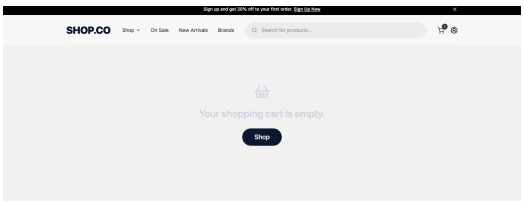

($a_1$) Sonnet, **before** click



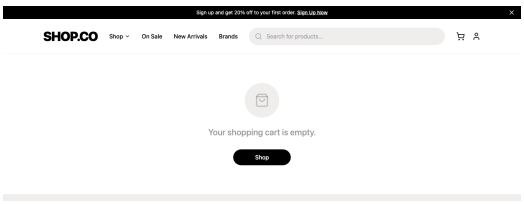

($a_2$) Sonnet, **after** wired: item, $145, badge **1**



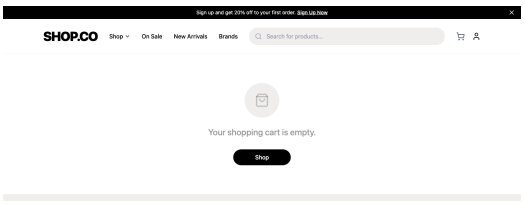

($b_1$) GPT-5.4, **before** click




($b_2$) GPT-5.4, **after** dead code: cart empty, badge **1**



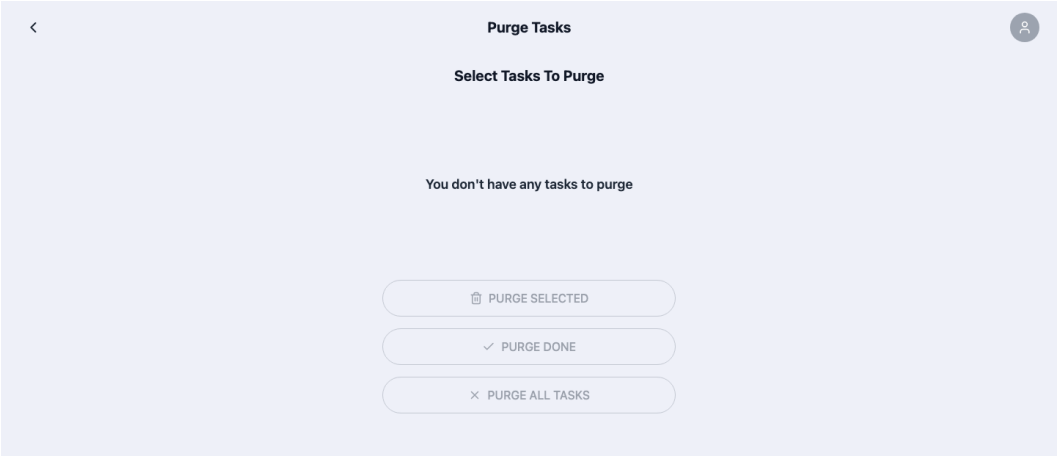

($c_1$) Gemini, **before** click



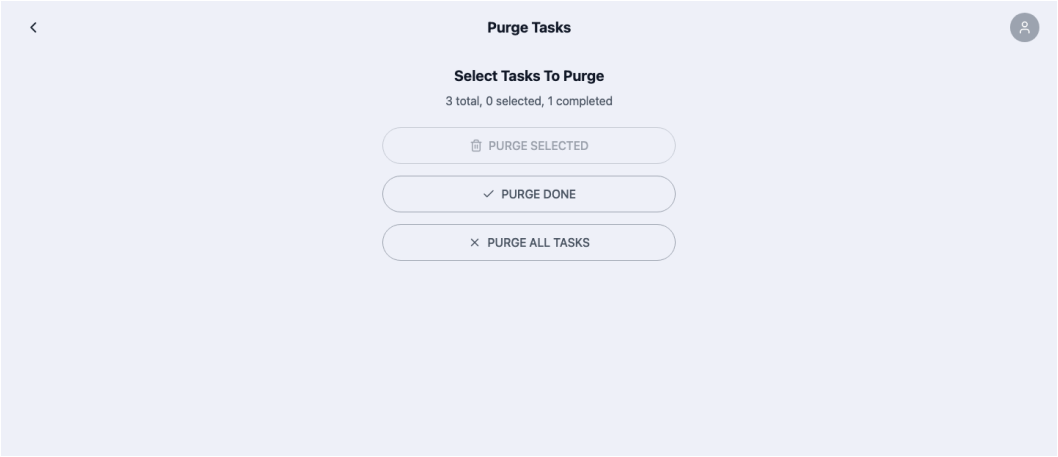

($c_2$) Gemini, **after** inert: no change, no badge

Figure 9: **State-machine inference on `48_shopco-ecommerce`: before/after live capture across an identical click sequence.** Each row is one model, before/after an identical add-to-cart → cart-navigation sequence (SPA `<Link>`, no hard reload, so in-memory state must survive). Left column (before): all three carts are empty baselines. Right column (after): ($a_2$) Sonnet's `CartContext` propagates end-to-end: GRADIENT GRAPHIC T-SHIRT (Size Large, $145) appears, subtotal recomputes, header badge reads **1**; ($b_2$) GPT-5.4's badge *also* reads **1** (`addToCart` did mutate `cartCount`), yet the cart route still renders "Your shopping cart is empty.": `CartPage` never reads `cartItems` from `ShopContext`, a literal dead-code branch; ($c_2$) Gemini's render is visually identical to its pre-click state (no badge change, no list item), no state container at all. All six panels (three models × before/after) are comparable in VFS, yet IIS on the S3 cross-route scope splits them sharply.

**Persistent task state (`39_maciekt07-todoapp`).** The todo task supplies 5 screenshots of a single-page task manager with categories, multi-select, completion toggles, and JSON import/export. Of the six models that generated this app, only Kimi K2.5 persists task state, lifting it into a single `AppContext` that mirrors itself into `localStorage` on every mutation (`STORAGE_KEY = "taskapp_data"`) and re-hydrates on mount. None of the other five survives a reload: Claude implements working in-memory CRUD in an app-level Context, while GPT-5.4, Gemini, and Qwen render the task list as static mock data with no mutable state; GLM-4.6V's generated source holds tasks in component-local `useState`, but the app fails to build ($EXEC@3$=F). Figure 10 verifies the re-hydration half directly: three tasks seeded into `localStorage` survive a hard reload and re-appear on mount. This cross-session (`localStorage`-backed) persistence is strictly stronger than the S3 cross-route criterion (§B), which several models satisfy only in memory (e.g., Sonnet's cart in Fig. 9).



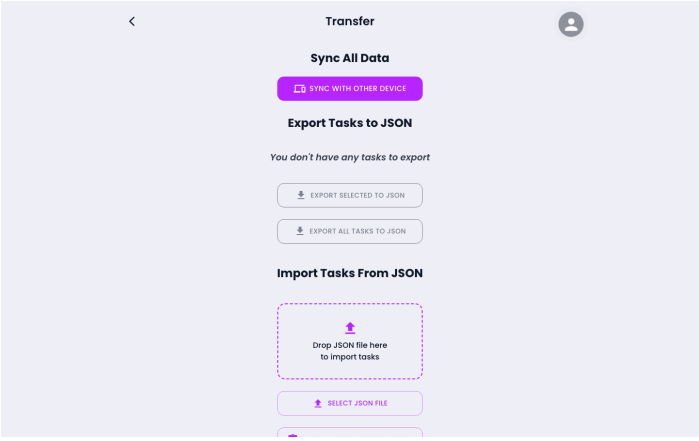

(a) Kimi `/purge`, fresh ctx "no tasks"



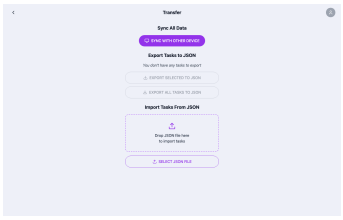

(b) Kimi `/purge`, after seeding `localStorage` + reload "3 total, 0 selected, 1 completed"

Figure 10: **Cross-session state persistence on `39_maciekt07-todoapp`.** (a) the `/purge` route on a fresh context shows "no tasks to purge" and disabled action buttons; (b) after seeding three tasks (one completed) into `localStorage['taskapp_data']` and triggering a hard reload, the same route reports "3 total, 0 selected, 1 completed" and re-enables the PURGE DONE/PURGE ALL TASKS buttons, proving that Kimi's `AppContext` re-hydrates from `localStorage` on mount. None of the remaining five baselines survives a reload: Claude holds tasks in an app-level Context (in memory only) while GPT-5.4, Gemini, and Qwen render the list as static mock data; GLM-4.6V's source uses component-local `useState` but the app fails to build.

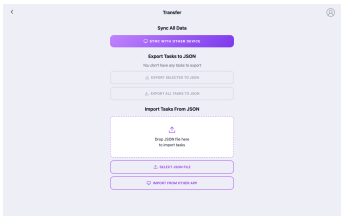
**Drag-and-drop file upload (`39_maciekt07-todoapp`).** The same task's `/transfer` screen shows a dashed-border "Drop JSON file here to import tasks" import region. The cue is purely visual: a dashed rectangle with an upload icon and a brief instruction label, with no API specification. Kimi K2.5 and Claude Sonnet 4.6 both extract a `FileDropZone` component bound to native HTML5 `onDragOver`/`onDrop` events that pre-fills the JSON-import field on drop. Gemini 3.1 Pro Preview, GPT-5.4, and Qwen3.5-397B-A17B render the dashed rectangle as decoration plus a click-only fallback button with no drop handlers; GLM-4.6V's build of this app fails ($EXEC@3$=F).



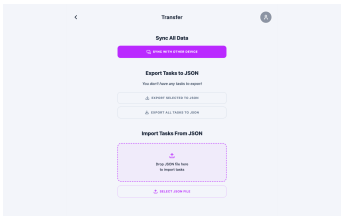

(a) input `/transfer`



(b) Kimi K2.5 onDrop wired



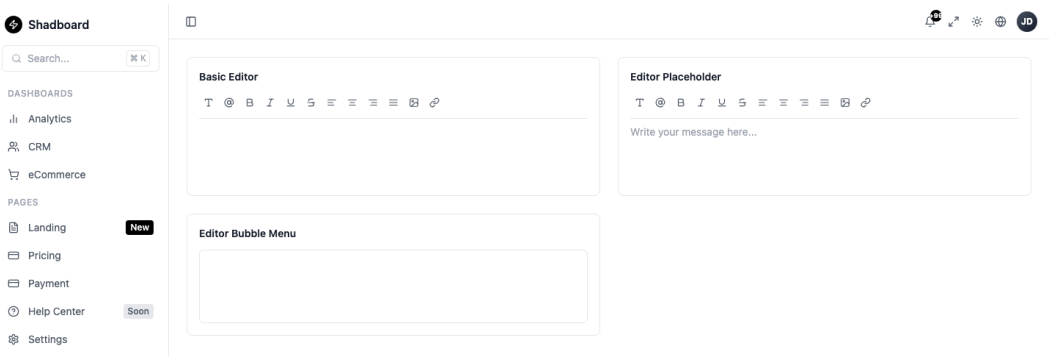
(c) Sonnet 4.6 onDrop wired



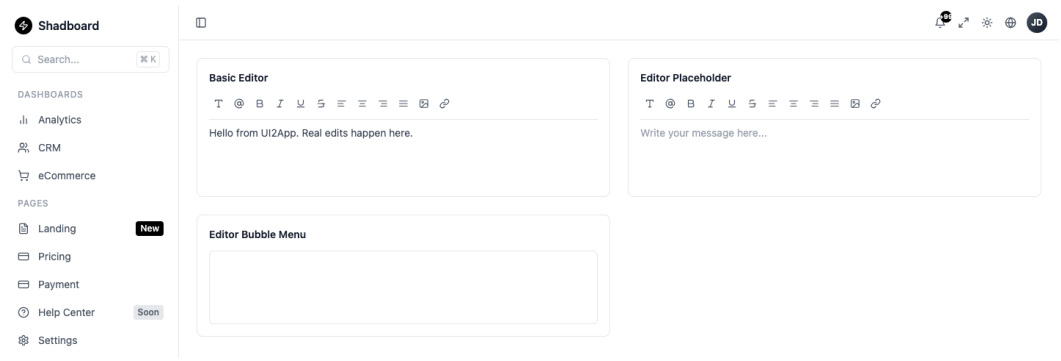
(d) Gemini 3.1 decoration only

Figure 11: **HTML5 DnD primitive reach on `39_maciekt07-todoapp`.** Kimi/Sonnet bind `onDragOver`+`onDrop` to a real file-import flow; Gemini produces the same rectangle as decoration. Identical visual targets, divergent event topology. Pairs with the audio case (Fig. 8) to illustrate that frontier models repeatedly reach for domain primitives invisible at the pixel level.

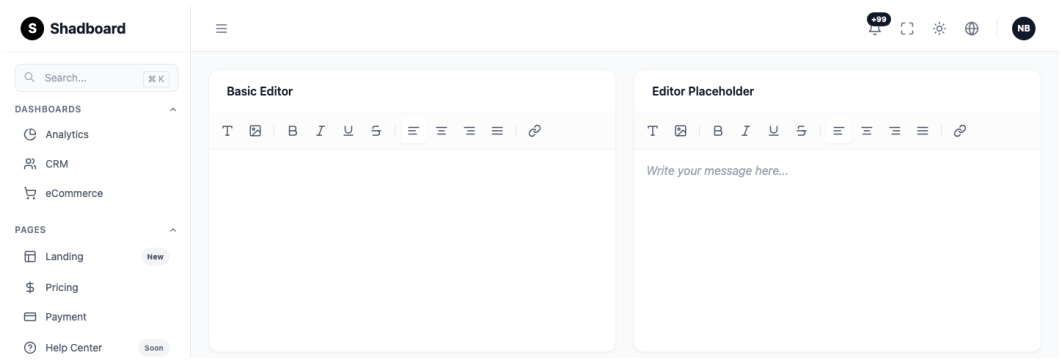
**Rich-text editing (`67_shadboard`).** `67_shadboard` ships an editor screen whose only behavioral cue is a formatting toolbar (bold, italic, list, link) above an empty document area. Claude Sonnet 4.6 and Qwen3.5-397B-A17B mark the document area with `contentEditable`, allowing genuine cursor-driven typing on the rendered page. Kimi K2.5 does the same and additionally binds the toolbar buttons to `document.execCommand`. GPT-5.4 is the sharper negative: it renders the full editor UI — a formatting toolbar above a document pane — that *looks* like a working editor, yet the pane is a static `<div>` (the toolbar icons carry no handler and there is no `<textarea>` or `contentEditable` element), so typing produces no change. The failure is indistinguishable from a real editor at idle and surfaces only under the live-typing test (Fig. 12). The remaining two models land between these poles: GLM-4.6V wires a plain `<textarea>` that accepts text but leaves its formatting toolbar inert, and Gemini 3.1 Pro Preview emits no `/editor` route at all.



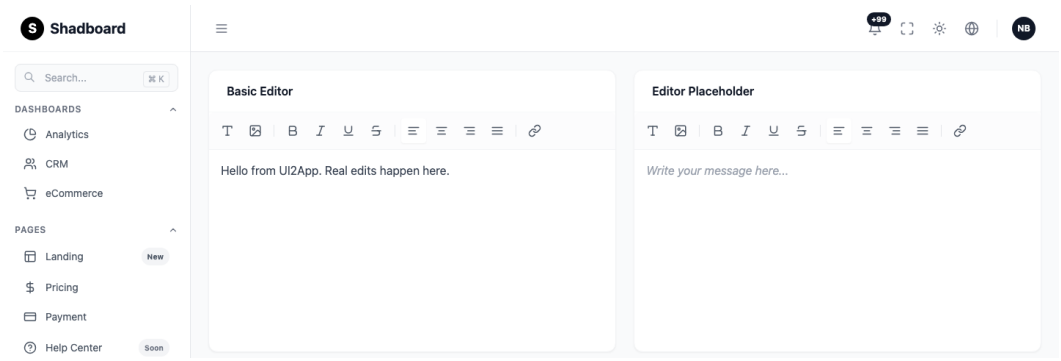

($a_1$) Sonnet `/editor`, **idle**



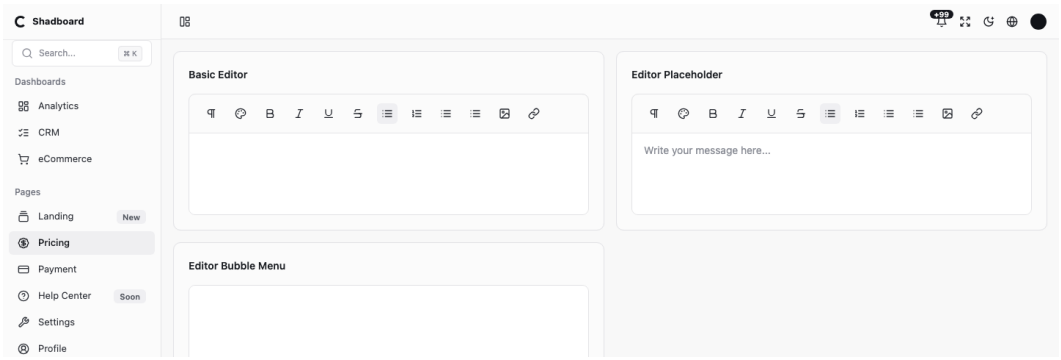

($a_2$) Sonnet, **after typing**: text appears



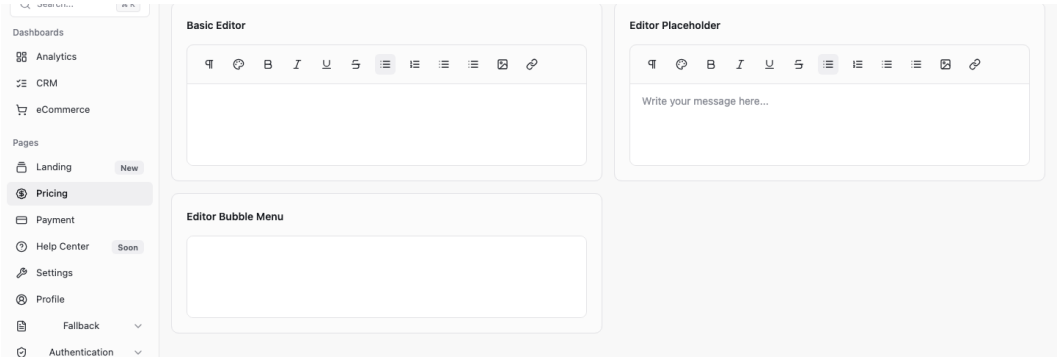

($b_1$) Qwen3.5 `/editor`, **idle**




($b_2$) Qwen3.5, **after typing**: text appears



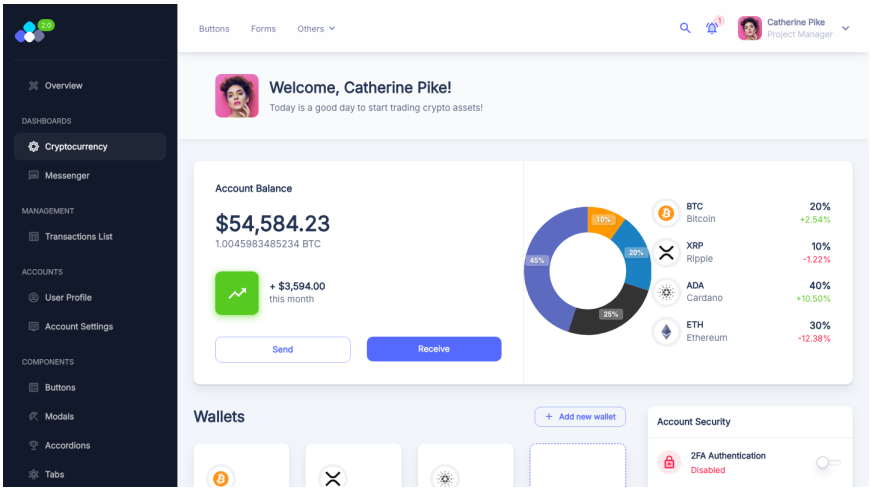

($c_1$) GPT-5.4 `/editor`, **idle**: looks like an editor



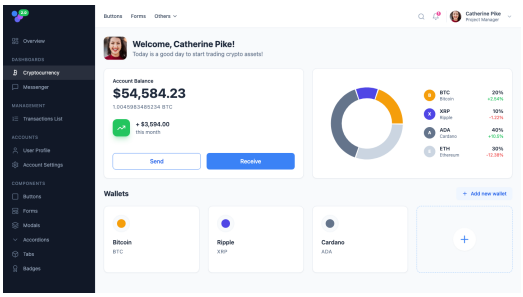

($c_2$) GPT-5.4, **after typing**: no change — static `<div>`, no editable element

Figure 12: **Editor primitive on `67_shadboard`: before/after typing into the document area.** Top of each pair (idle): each model immediately after navigating to its editor route (Sonnet: 3 `[contenteditable]` regions detected; Qwen: 1; GPT-5.4: 0 — the document panes are static `<div>`s under a decorative toolbar). Bottom (after typing): the same test string is typed into the first document pane. ($a_2$, $b_2$) Sonnet and Qwen show the typed text inside the Basic Editor card — the toolbar+document affordance wired into a real editing surface. ($c_1$, $c_2$) GPT-5.4 renders a visually faithful editor whose panes are inert: the after-typing capture is identical to idle (no text appears). The live-typing test exposes the divergence IIS captures and pixel fidelity cannot.

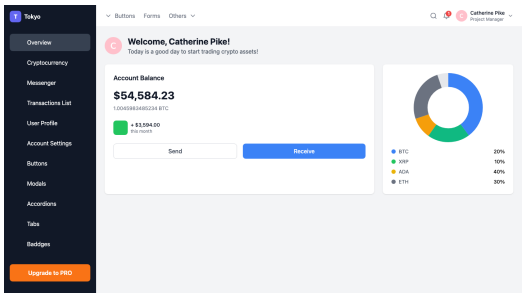
**Reference-subset rendering (`62_tokyo-mui-dashboard`).** The Tokyo crypto-dashboard organizes content into two vertical sections: a top section (Welcome header, ACCOUNT BALANCE card, donut chart with per-asset breakdown) and a continuation (WALLETS row of three asset cards plus the ACCOUNT SECURITY side card). Gemini 3.1 Pro Preview reproduces both sections: the WALLETS header is visible at the bottom of the captured viewport, with the first row of asset-card frames already laid out. GLM-4.6V's render is a *strict subset* of the reference: the top section is reproduced faithfully (the donut, the ACCOUNT BALANCE card with the same numeric value and Send/Receive controls, the per-asset legend), while the WALLETS row and ACCOUNT SECURITY card have *no counterpart* in the generated DOM. We verify the subset relation rather than infer it from a viewport screenshot: a full-page Playwright capture across seven candidate routes (`/dashboard`, `/crypto`, `/cryptocurrency`, `/dashboards/crypto`, `/dashboard-crypto`, `/overview`, `/`) returns `document.documentElement.scrollHeight=900px` (exactly the viewport height, so no overflow exists where missing blocks could hide), and `document.body.innerText` on every probed route contains none of the WALLETS/SECURITY section keywords. The Wallets/Security blocks are absent from the rendered DOM, not pushed past the viewport.




(a) input `/crypto`: top row + WALLETS header + ACCOUNT SECURITY card visible




(b) Gemini 3.1 below-fold preserved: top row + WALLETS row




(c) GLM-4.6V strict reference subset: top section only; Wallets/Security blocks have no DOM counterpart (`scrollHeight=900`, probed keywords absent)

Figure 13: **Reference-subset rendering on `62_tokyo-mui-dashboard/crypto`.** Pictorial counterpart to §E. GLM-4.6V's rendered DOM is a strict subset of the reference dashboard's DOM: the top section is reproduced faithfully, while the reference's subsequent WALLETS+SECURITY blocks have no counterpart in the generated DOM. We verify the subset relation, not just a viewport screenshot: a full-page Playwright capture across seven candidate routes returns `scrollHeight` = 900 px (no overflow) and `innerText` contains none of the WALLETS/SECURITY section keywords. Gemini preserves both sections.

**Build-time failures (`55_shadcn-nextjs-dashboard`, `60_mantine-dashboard`).** The two dominant EXEC@3 modes (Table 10) have orthogonal etiologies. C1 (hallucinated icon) is a factuality error: GPT-5.4 imports `Funnel` from `lucide-react`, but in the scaffold-pinned `0.460.0` the icon is still exported as `Filter` (`Funnel` is the post-`0.460` canonical name); `vite build` aborts with "`"Funnel" is not exported by .../lucide-react.js`". C2 (`package.json` overwrite) is an instruction-compliance error: GLM-4.6V regenerates the scaffold's `package.json` despite the explicit "Do NOT include scaffold files" clause (Appendix H, clause iv), pinning the unpublished version `"mantine":"^1.0.0"`; `pnpm install` aborts with `ERR_PNPM_NO_VERSIONS`.

**AnalyticsPage.tsx**

```
import {
  Download, CalendarDays,
  Funnel              // <-- not
    exported
} from "lucide-react";
```

Listing 3: (a) C1, GPT-5.4 on `55_shadcn-nextjs-dashboard`: hallucinated `lucide-react` export. Factuality error.

**package.json (regenerated)**

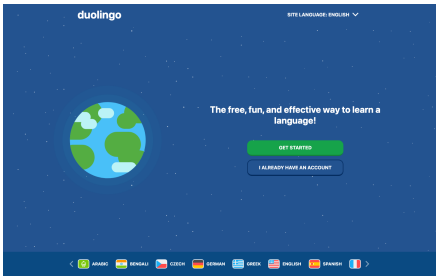
```
{
  "dependencies": {
    "mantine": "^1.0.0"    // pkg name is
                           // @mantine/
      core
  }
}
```

Listing 4: (b) C2, GLM-4.6V on `60_mantine-dashboard`: regenerated `package.json` (clause-iv violation). Instruction-compliance error.

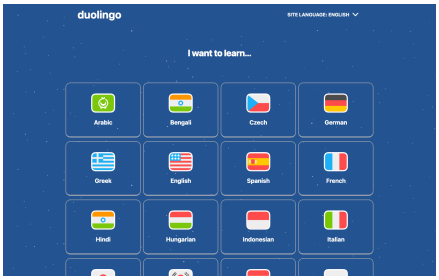
Figure 14: **Factuality vs instruction-compliance failures.** C1 (left) is a symbol-existence hallucination that the prompt does not pre-empt. C2 (right) violates clause (iv) of the plan-step prompt. The orthogonality of the two error modes motivates reporting an instruction-compliance rate alongside EXEC@3 (§G.1).

**Functional quiz engine (`61_react-duolingo`).** The gamified learning app's 10-screenshot input set (Fig. 15) spans a lesson map, a lesson exercise, a leaderboard, a shop, a profile, and settings. The lesson exercise — a prompt, three answer tiles, and a CHECK button — has behavior the screenshots never state: selecting a tile, judging it against the correct answer, and awarding XP on success must all be inferred. Across the six baselines the rendered exercise is near-identical,

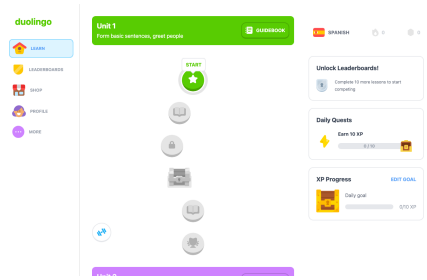
yet only Claude Sonnet 4.6 wires a working quiz engine — `handleCheck` compares the selection to `question.correctId`, sets correct/incorrect feedback, and on a correct answer calls `incrementLessons` (advancing lesson progress and adding 10 XP). Kimi K2.5 only partially wires it (its CHECK handler navigates without validating or scoring), and the remaining four (Gemini 3.1 Pro Preview, GPT-5.4, Qwen3.5-397B-A17B, GLM-4.6V) render a static mockup whose CHECK button carries no handler. The 1-functional / 1-partial / 4-static split is invisible to VFS — every render shows the same tiles and button — and is the answer-validation analogue of the audio case (Fig. 8).



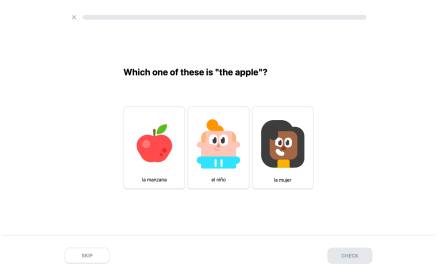

/



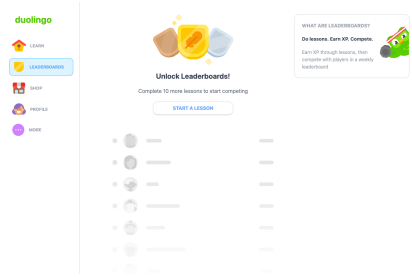

/register



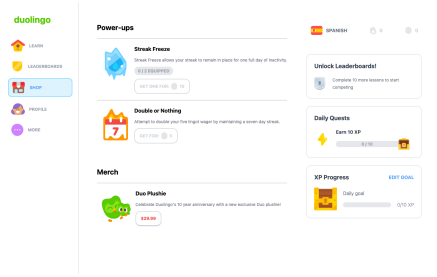

/learn



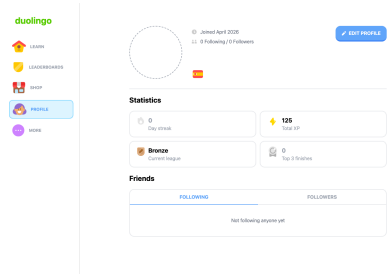

/lesson



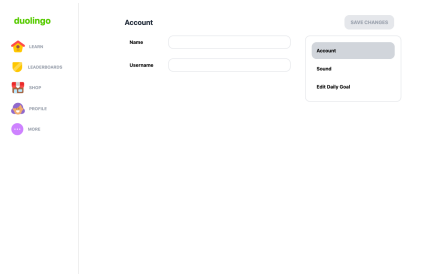

/leaderboard



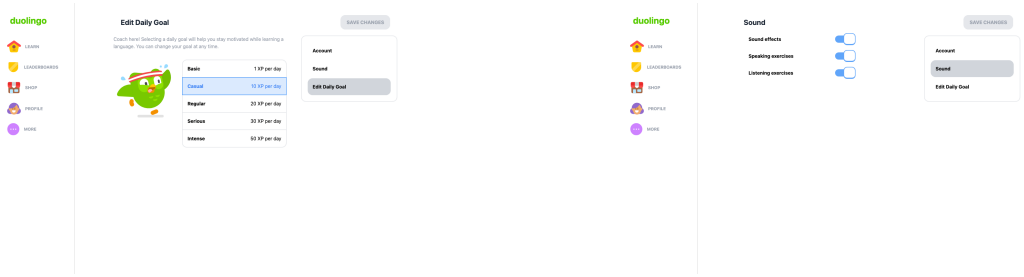

/shop




/profile

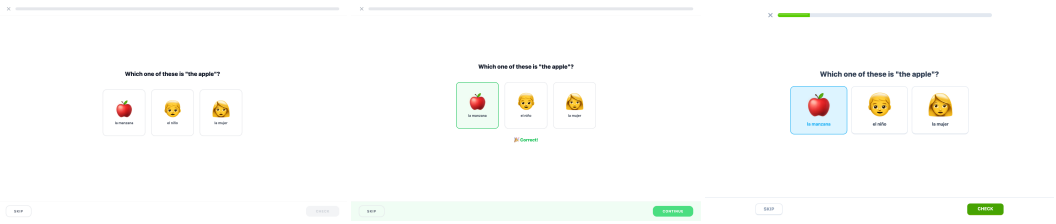



/settings/account

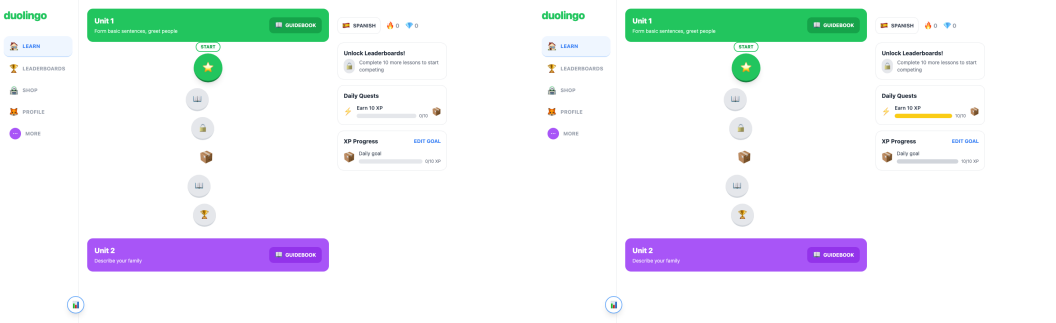


`/settings/coach` `/settings/sound`

Figure 15: **Full input set for `61_react-duolingo` (10 screenshots, one per route)** given to every model. The lesson exercise (`/lesson`) implies an answer-validation + XP-scoring engine; whether each model actually builds it is shown in Fig. 16.



(a) Claude lesson — **before** (b) Claude — **after**: *Correct!* + CONTINUE (c) Gemini — **after**: inert, no feedback



(d) Claude `/learn` — **before**: Daily Quest 0/10 XP (e) Claude `/learn` — **after**: Daily Quest 10/10 XP

**LessonPage.tsx**

```tsx
const handleCheck = () => {
  if (!selected) return;
  const ok = selected === question.correctId;
  setCorrect(ok); setChecked(true);
  if (ok) incrementLessons();  // +1 lesson, +10 XP
};
```

Listing 5: Claude — `LessonPage.tsx` (extract): the answer-validation + XP engine the other five omit.

Figure 16: **Functional quiz engine on `61_react-duolingo` (before/after the same select-answer→CHECK sequence).** (a,b) Claude validates the answer and shows *Correct!* feedback, advancing to CONTINUE; (c) Gemini's CHECK has no handler, so the same clicks leave the question unchanged. (d,e) the correct answer also awards XP, filling Claude's daily quest $0/10 \rightarrow 10/10$. Only Claude wires the full engine (Kimi partial; the other four inert) — an answer-validation + scoring affordance VFS cannot see, since every render is near-identical.

**Synthesis across cases.** The cases illustrate differences in the interaction types implemented by the evaluated models. Kimi K2.5 implements audio synthesis and cross-session task persistence, while Claude Sonnet 4.6 implements the demonstrated cart and quiz interactions. Build failures also differ across models: GPT-5.4's failures are dominated by hallucinated icon exports, whereas GLM-4.6V's are dominated by scaffold-related instruction violations (Table 10). These observations complement the aggregate scores by showing which behaviors and implementation failures underlie them.

## H PROMPT TEMPLATES

Below are the prompt templates used in the pipeline. Placeholders rendered as `{name}` are populated at run time.

### STAGE 1 (PLAN).

The Plan prompt is issued in the second user turn after the assistant has emitted the scaffold listing in Turn 1. The same turn carries all $M$ screenshots as image content.

```
You are an expert frontend developer.  You are given {M} screenshots
of a web application.  Your task is to reproduce the UI as a complete,
runnable Vite + React + TypeScript + Tailwind CSS project and
implement the interactions implied by the screenshots.

The project scaffold has already been set up (you saw it above).
src/App.tsx is the root component mounted by main.  All source files
you generate must be under src/.

tsconfig uses "jsx":  "react-jsx" so explicit React imports are
unnecessary.  react-router-dom and lucide-react are available in
dependencies.

<code_guidelines>
- Use coding best practices.
- Use Tailwind CSS utility classes for styling.  Match the screenshots
as closely as possible.
- Use relative imports.  Do not add file extensions in imports for
.ts/.tsx files.
- Files containing JSX must use .tsx extension.
</code_guidelines>

Analyze the {M} screenshots carefully.  Then output a JSON file
plan -- a list of all source files you will create, with a one-line
description of each file's purpose.

Output ONLY valid JSON (no markdown fences, no explanation):

{"extra_dependencies":  {}, "plan":  [{"path":  "src/App.tsx",
"description":  "Root component with routing setup"}, ...]}

Rules:
- Include src/App.tsx and all component/page files.  All paths start
with "src/".
- "extra_dependencies" lists any npm packages beyond the scaffold.
- Do NOT include scaffold files.  Do NOT generate any code yet -- only
the plan.
```

The four constraints we treat as explicit instruction clauses for the failure-mode attribution in §4.4 are: (i) all source files under `src/`; (ii) relative imports; (iii) every non-scaffold package declared in `extra_dependencies`; (iv) no regeneration of scaffold files.

### STAGE 2 (GENERATE).

The Generate prompt is issued in the next user turn after the model has returned the plan. Screenshots are not re-sent.

```
Now generate the COMPLETE source code for ALL files in the plan below.

<project_plan>
{plan_summary}
</project_plan>

Output each file using this exact format (no markdown fences, no
explanation):

-- src/App.tsx --
[complete source code for App.tsx]

-- src/components/Navbar.tsx --
[complete source code for Navbar.tsx]

...and so on for every file in the plan.

CRITICAL RULES:
- Write COMPLETE file content for every file.  Never truncate,
abbreviate, or use comments like "// rest of code".
- Files containing JSX/TSX syntax (<div>, <Component/>) MUST use .tsx
extension.
- Use 2 spaces for indentation.
- Every file must start with "-- filepath --" on its own line.
- Do NOT output anything other than the file blocks.
```

If the model's response is truncated (`finish_reason` $\neq$ `stop`), we send up to three Continue turns: "Continue writing from exactly where you left off. Do not repeat any content. Continue with the next file or finish the current file."

STAGE 3 (SELF-DEBUG).

On EXEC failure the same model is re-invoked for up to three rounds. Each round runs three sub-prompts as needed.

**(a) File-localization fallback.** When the regex extractor cannot identify which source files to repair from the build error, the model is asked to localize:

```
The following build error occurred.  Which source file(s) need to be
fixed?

<error>
{errors}
</error>

<project_files>
{file_list}
</project_files>

Reply with ONLY the file path(s) that need fixing, one per line.  No
explanation.
```

**(b) Build-error repair.** For each identified file the model receives the error, the project file listing, and the file's current contents:

```
Fix the following build error in {filepath}:

<error>
{errors}
</error>

<project_files>
{file_listing}
</project_files>
```

```
<current_code file="{filepath}">
{current_code}
</current_code>

Output ONLY the fixed complete file content -- no explanation, no
markdown fences.
```

**(c) Runtime-error repair.** When the build passes but the home route fails to render, the model receives all candidate files (`App.tsx`, `main.tsx`, plus any flagged by error parsing) together for a holistic fix:

```
The React app builds successfully but renders a blank page at runtime.

<error>
{errors}
</error>

{file_contents}

For EACH file that needs changes, output:
-- filepath --
<complete fixed file content>

Output ONLY the fixed files -- no explanation, no markdown fences.
```

# I AUTOMATED IIS EVALUATION

**Evaluator and human validation.** GPT-6 Luna with medium reasoning effort evaluates generated source code and rendered screenshots without access to the generator identity, assigning coverage, outcome, and scope labels under the interaction rubric. IIS uses the fixed reference inventory and Eq. 1. We validate the evaluator against human annotations on a 45-application subset: 217 executable artifacts yield 1,519 category-level comparisons; 53 non-executable artifacts receive zero IIS. Coverage agreement is $86.4\%$. On the 868 jointly applicable items, agreement on interaction outcomes (working, partial, or failed) is $81.8\%$ and exact scope agreement is $91.1\%$.

Table 11: Human and GPT-6 Luna IIS on the 45-application validation subset. Differences use unrounded model means.

| Model | Human IIS | Auto IIS | Auto − Human |
|---|---|---|---|
| Claude Sonnet 4.6 | 36.46 | 37.30 | +0.84 |
| Gemini 3.1 Pro Preview | 7.30 | 7.87 | +0.56 |
| Kimi K2.5 | 18.80 | 22.57 | +3.78 |
| GPT-5.4 | 6.76 | 5.20 | -1.56 |
| Qwen3.5-397B-A17B | 12.21 | 13.02 | +0.80 |
| GLM-4.6V | 3.36 | 1.47 | -1.89 |

Across the six model means, Spearman correlation is $1.00$, Pearson correlation is $0.991$, MAE is $1.57$ IIS points, and mean signed bias is $+0.42$ points. Automated IIS estimates implementation correctness from source and screenshots; the validation labels are based on observed runtime behavior.

# J REPEATED-GENERATION EVALUATION

To assess generation variability, each application in the 45-app subset is generated independently three times by each of six models, yielding 810 application–model–generation outcomes. GPT-6 Luna (medium) evaluates all 644 executable artifacts using the same rubric and fixed RII; the remaining 166 outcomes receive zero VFS and IIS. The mean and sample standard deviation in Table 12 are computed from three round-level 45-application model means ($\mathrm{ddof} = 1$).

Table 12: Three-generation evaluation on the 45-application subset. Values are the mean ± sample standard deviation of the three round-level model means. IIS uses GPT-6 Luna (medium), as in Table 2.

| Model | EXEC@3 (%) | VFS | IIS |
|---|---|---|---|
| Claude Sonnet 4.6 | $98.52 \pm 1.28$ | $69.55 \pm 5.33$ | $36.86 \pm 0.39$ |
| Gemini 3.1 Pro Preview | $98.52 \pm 1.28$ | $71.68 \pm 5.64$ | $8.95 \pm 3.55$ |
| Kimi K2.5 | $76.30 \pm 8.98$ | $54.48 \pm 9.33$ | $18.73 \pm 3.34$ |
| GPT-5.4 | $85.19 \pm 2.57$ | $60.37 \pm 4.04$ | $3.97 \pm 1.68$ |
| Qwen3.5-397B-A17B | $77.78 \pm 0.00$ | $54.13 \pm 1.64$ | $14.64 \pm 1.91$ |
| GLM-4.6V | $40.74 \pm 4.63$ | $24.27 \pm 2.63$ | $2.77 \pm 2.03$ |

**Hierarchical uncertainty.** We use 20,000 hierarchical bootstrap replicates (seed 20260925). Each replicate samples 45 applications with replacement, preserving application pairing across models, then independently resamples three generations within each selected application–model cell. The 95% percentile interval for Claude minus Gemini IIS is $[20.95, 35.06]$, around a mean difference of $27.91$ points. Claude leads IIS and Gemini leads VFS in every round. This interval captures application composition and generation variability, not evaluator error.

Table 13: Round-level results on the 45-app subset. Within each entry, values are R1 / R2 / R3.

| Model | EXEC@3 (%) | VFS | IIS |
|---|---|---|---|
| Claude Sonnet 4.6 | 100.00 / 97.78 / 97.78 | 75.68 / 67.03 / 65.95 | 37.30 / 36.54 / 36.76 |
| Gemini 3.1 Pro Preview | 100.00 / 97.78 / 97.78 | 78.14 / 67.71 / 69.19 | 7.87 / 6.07 / 12.92 |
| Kimi K2.5 | 86.67 / 71.11 / 71.11 | 65.25 / 48.98 / 49.22 | 22.57 / 17.13 / 16.50 |
| GPT-5.4 | 82.22 / 86.67 / 86.67 | 64.96 / 58.80 / 57.36 | 5.20 / 2.06 / 4.65 |
| Qwen3.5-397B-A17B | 77.78 / 77.78 / 77.78 | 56.02 / 53.21 / 53.16 | 13.02 / 14.17 / 16.75 |
| GLM-4.6V | 35.56 / 44.44 / 42.22 | 21.28 / 25.32 / 26.22 | 1.47 / 1.74 / 5.11 |