REFCRAFT: BENCHMARKING VISUAL-REFERENCE-CONDITIONED AGENTIC GENERATION THROUGH WEB DEVELOPMENT

Anonymous authors
Paper under double-blind review

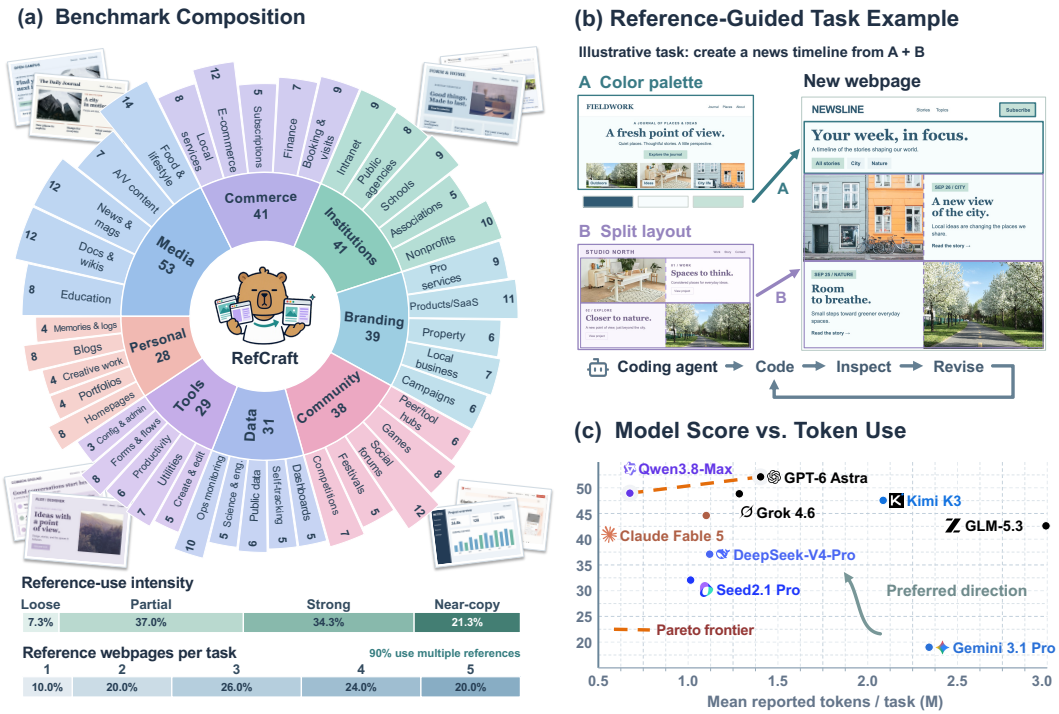


Figure 1: Overview of RefCraft. (a) RefCraft spans a broad spectrum of practical web development domains and features visual references that vary in intensity and source. (b) An example task in which a new webpage borrows the color palette from source webpage A and the layout from source webpage B. The agent may iteratively render, inspect, and revise the webpage. (c) Overall score versus reported token usage for nine frontier models. The dashed line denotes the Pareto frontier. The results show that the task remains far from solved.

ABSTRACT

Beyond basic tool use and coding skills, effectively incorporating visual references into agentic generation is equally essential for applying AI agents to real-world productivity tasks. We study this crucial capability through webpage development, a practical and readily evaluable setting in which existing benchmarks remain largely idealized: they typically focus on reconstructing visual targets or generating webpages from textual instructions alone, overlooking the more common workflow of developing from visual examples. Specifically, we introduce RefCraft, a benchmark comprising 300 realistic multimodal webpage-development tasks that combine natural-language instructions with screenshots from one to five reference webpages. Agents must selectively interpret and utilize these references while rendering, visually inspecting, and iteratively refining their outputs. We evaluate the resulting open-ended artifacts using independently verifiable keypoints and complementary programmatic and VLM judges. Across nine frontier (M)LLMs, the strongest system achieves an overall score of only 52.12 out of 100, revealing that

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.

visual-reference-conditioned agentic generation remains a substantial challenge for current models in practical webpage-development settings.

1 INTRODUCTION

Recently, as coding and co-working assistants, (M)LLMs have transformed numerous fields and substantially improved productivity (OpenAI, 2026b; Anthropic, 2026a; Qwen Team, 2026b; Kimi et al., 2026). This success has further fueled growing enthusiasm among researchers and practitioners for extending large models to an even broader range of practical tasks, as exemplified by GPT-6 Astra (OpenAI, 2026d) and Claude Fable 5.1 (Anthropic, 2026b). When we scrutinize some common productivity tasks, we find that **visual references** play an important role: designers may first study successful examples before developing new proposals and then continuously refine their designs through visual inspection, while CAD engineers examine existing parts to design new ones that fit and function properly. This observation highlights that, for both human experts and AI agents, *basic tool use and coding skills alone do not encompass the full spectrum of productive capabilities—effectively **incorporating visual reference and inspection** into agentic generation is equally essential.*

To evaluate this important yet subtle capability, we ground it in a focused and readily verifiable testbed: webpage development, which is ubiquitous in everyday digital life and produces highly visual, easily assessable outputs. In fact, a growing body of benchmarks has investigated general webpage-related tasks, but most either use natural-language instructions without visual references (Lu et al., 2026; Zhu et al., 2025) or focus solely on reconstruction tasks, such as reconstructing webpages from screenshots (Yun et al., 2024; Wang et al., 2026) or videos (Chen et al., 2026). More recent studies have broadened the evaluation setting by incorporating interactive operations (Xiao et al., 2025a) or introducing webpage editing and repair tasks (Lei et al., 2026). Nevertheless, we argue that **these prior efforts remain academically idealized and differ considerably from common real-world development workflows**. In practice, frontend developers and users often begin with a rough design idea and browse existing websites for inspiration. For instance, a user may adopt the appealing layout of a reference website while redesigning its fonts, content, and other elements for a new purpose. Fully reconstructing an existing website is a far less common or representative task. To bridge this gap and, more importantly, evaluate whether AI agents can effectively conduct such a visual-reference-conditioned agentic generation process, we introduce RefCraft.

RefCraft comprises 300 diverse and realistic multimodal queries spanning a broad range of web development scenarios. Each query combines a natural-language user instruction with screenshots from one to five reference webpages, and models solve the task in an inspect-and-iterate setting where they can render and refine intermediate outputs. As shown in Fig. 1, RefCraft varies substantially in domains, reference composition, and the intended degree of reference utilization—from closely following a reference to selectively adapting its overall style or specific local elements. Evaluating such tasks requires moving beyond conventional target-based evaluation: a query may admit multiple valid solutions, and visual similarity alone cannot determine whether the references have been used appropriately. We therefore decompose each complex and subtle query into a verified set of key points, turning open-ended requirements into independently assessable criteria. Moreover, webpages are both visual and interactive artifacts, encompassing subjective perceptual qualities as well as objective executable behaviors. We thus adopt a dual-track evaluation protocol that uses programmatic judges for deterministic requirements and a VLM judge for perceptual and semantic ones. Together, these designs enable reliable evaluation of whether a generated webpage fulfills the user's intended use of visual references, rather than merely resembling them.

We benchmark 9 frontier (M)LLMs on RefCraft under a unified harness. Despite access to visual references and an inspect-and-iterate workflow, the best-performing system, GPT-6 Astra (OpenAI, 2026d), achieves an overall score of only 52.12, leaving substantial room for improvement. Component-level results reveal that models often satisfy many individual requirements but fail to realize an essential reference relationship or core behavior. To assess the reliability of RefCraft, we compare its model ranking with both our expert human study and the human-preference leaderboard of Arena.AI[1], finding strong agreement with both. These findings indicate that current agents can reproduce salient visual attributes but still struggle to translate instruction-dependent reference relationships into coherent designs—precisely the capability that matters to human users.

[1] https://arena.ai/leaderboard/code/image-to-webdev

2 Related Work

2.1 Visual-Reference-Conditioned Agentic Generation

In recent years, we have witnessed rapid advances in AI agents. From early frameworks such as ReAct (Yao et al., 2022) to coding agents such as SWE-agent (Yang et al., 2024) and OpenHands (Wang et al., 2025a), substantial effort has been devoted to reasoning, code generation, long-horizon planning, and tool use (Schick et al., 2023; Wang et al., 2024; Yang et al., 2026). In comparison, vision has received less attention in agentic generation and is typically used only as a perceptual interface for interpreting image or video observations (Xie et al., 2024; Qin et al., 2025; Chen et al., 2024). Meanwhile, visual inputs serve as important conditioning signals in generative models, particularly diffusion-based models for image and video generation (Seedance et al., 2026; Bai et al., 2025; Wang et al., 2025b). Yet methods ranging from DreamBooth (Ruiz et al., 2023) and ControlNet (Zhang et al., 2023) to IP-Adapter (Ye et al., 2023) and the OmniGen series (Xiao et al., 2025c; Wu et al., 2025) often assign predefined, coarse-grained roles to visual references—for example, preserving subject identity, transferring overall style, or imposing structural constraints. Consequently, both prominent research directions leave a challenging yet practical setting underexplored: visual-reference-conditioned agentic generation, where an agent must interpret subtle user instructions and selectively incorporate heterogeneous visual references throughout the agentic generation process.

2.2 Web Development Benchmarks

Existing web development benchmarks broadly follow two task formulations. Text-driven benchmarks, such as WebGen-Bench (Lu et al., 2026) and FrontendBench (Zhu et al., 2025), ask models to construct webpages or applications from natural-language specifications (Tran et al., 2026; Xu et al., 2025). Visual-target benchmarks instead formulate generation as reconstruction, including screenshot-to-code tasks such as Design2Code (Si et al., 2025) and Web2Code (Yun et al., 2024), video- or interaction-conditioned generation (Chen et al., 2026; Xiao et al., 2025a), and sketch-to-code generation (Jain et al., 2019). More recent benchmarks, including DesignBench (Xiao et al., 2025b), ArtifactsBench (Zhang et al., 2025), and WebCompass (Lei et al., 2026), extend beyond one-shot generation to interactive behavior, editing, repair, and multi-turn refinement (Yang et al., 2025; Sun et al., 2025; Saxena et al., 2026). Evaluation for these tasks has evolved accordingly, moving from visual-similarity measures based on CLIP features (Radford et al., 2021), layout blocks, colors, and positions (Wang et al., 2026; Si et al., 2025) toward MLLM- or agent-as-a-judge evaluation over rendered webpages (Zhang et al., 2025; Lei et al., 2026; Zhuge et al., 2024). Nevertheless, existing benchmarks typically treat visual input as a target to reconstruct or an artifact to edit. They rarely capture the common workflow in which users consult one or more webpages as partial design references, which is precisely the setting studied in our work.

3 Benchmark Design

Fig. 2 presents the overall design of RefCraft. Each benchmark instance, corresponding to a single task, consists of a textual user query, one to five visual references, and a verified set of evaluation keypoints. We construct these instances in three stages: curating a diverse reference pool (§ 3.2), synthesizing controllable user queries (§ 3.3), and decomposing each query into independently verifiable keypoints (§ 3.4). Further implementation details are provided in Appendix B.

3.1 Task Formulation

Each task t in RefCraft consists of a natural-language instruction q_t and a set of visual references $\mathcal{R}_t = \{R_i\}_{i=1}^{N_t}$, where $1 \leq N_t \leq 5$ and each R_i contains screenshots sampled from a reference webpage. Together with the evaluation keypoints $\mathcal{K}_t$, they define the task $\tau_t = (q_t, \mathcal{R}_t, \mathcal{K}_t)$. Model m executes the task and produces $h_{m,t} = \text{Agent}_m(q_t, \mathcal{R}_t)$, a self-contained HTML webpage, without access to $\mathcal{K}_t$. During generation, the agent may render its intermediate implementation, visually inspect the result, and iteratively revise the webpage. Unlike screenshot reconstruction, the objective is not to reproduce any reference exactly. Depending on the instruction, the agent may need to closely follow one reference, combine properties from multiple references, adapt selected design elements,

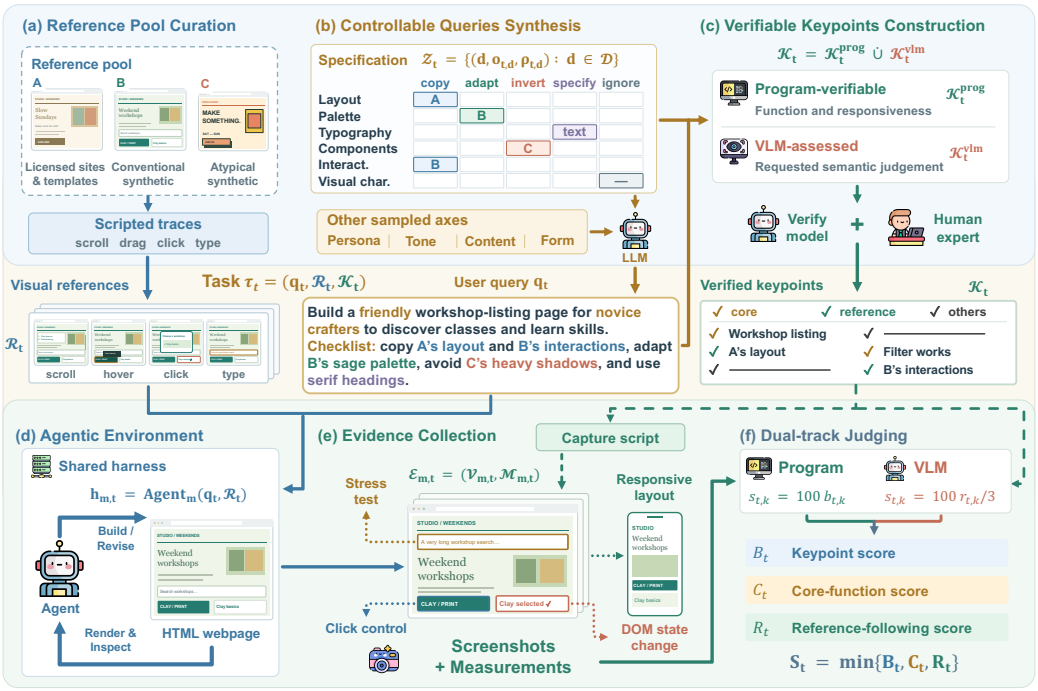


Figure 2: Construction and evaluation of RefCraft. (a) Reference webpages are captured across interaction states. (b) A specification spanning six visual dimensions and five requirement types guides natural-language query synthesis. (c) Task requirements are decomposed into program-verifiable and VLM-assessed keypoints. (d) Given the query and reference images, agents iteratively build, render, inspect, and revise a webpage in a shared harness. (e) Keypoints guide evidence collection from generated artifacts. (f) Complementary programmatic and VLM judges produce the base keypoint score B, core-function score C, and reference-following score R.

or deliberately avoid particular characteristics. The task therefore evaluates whether an agent can interpret and appropriately utilize visual references within an inspect-and-iterate workflow.

3.2 Reference Pool Curation

Reference webpages are important because they determine what information an agent can draw upon and therefore form the foundation of RefCraft. We curate the reference webpage pool according to three principles: **diversity**, covering a broad range of topics and design styles; **visual richness**, providing sufficiently salient properties for users to reference and evaluators to verify; and **realism**, reflecting designs that may plausibly appear in practical development workflows. Following these principles, we combine three sources: **(1)** openly licensed websites and webpage templates; **(2)** conventionally styled webpages synthesized by multiple models to increase diversity; and **(3)** deliberately synthesized webpages that deviate from conventional modern styles and common model-generated patterns. The rationale is that synthesized pages may inherit common model priors, allowing an agent to produce superficially similar designs based on its prior knowledge without consulting the references. To mitigate this shortcut, the third group incorporates less conventional visual characteristics, such as unusual color palettes, uncommon font combinations, and hard-edged shadows.

We represent each reference using images rather than source code or a live URL, preventing implementation details from leaking to the evaluated agent and better reflecting practical scenarios. Because a single static screenshot cannot expose dynamic properties such as hover effects or interactive state changes, we use Playwright[2] to capture a sequence of visual frames along scripted interaction traces involving scrolling, hovering, clicking, typing, and key presses. To support subsequent task construction, we also profile the observable information in each reference along six dimensions: *layout*, *palette*, *typography*, *components*, *interaction*, and *overall visual character*. Each dimension is annotated as *rich*, *weak*, or *none*, together with a concise description of the observed evidence. This profile prevents later queries from referring to properties that are absent or visually ambiguous.

[2] https://playwright.dev/

3.3 Controllable Query Synthesis

Directly prompting an LLM to generate user queries leads to substantial mode collapse: the resulting instructions are often similarly phrased and repeatedly emphasize only a small subset of the available reference information. We instead adopt a **specification-first** approach: we first determine the intended reference usage and task composition and then verbalize them as a natural user request.

Reference-use specification. As shown in Fig. 2(b), let $\mathcal{D}$ denote the six visual dimensions above. We represent the reference-use specification for task t as $\mathcal{Z}_t = \{(d, o_{t,d}, \rho_{t,d}) : d \in \mathcal{D}\}$. For each dimension, $o_{t,d} \in \{\textit{copy}, \textit{adapt}, \textit{invert}, \textit{specify}, \textit{ignore}\}$ specifies the requirement type, while $\rho_{t,d}$ identifies the assigned reference, with $\rho_{t,d} = \varnothing$ for *specify* and *ignore*. The five types are *copy*, which closely follows the reference; *adapt*, which preserves only selected characteristics; *invert*, which deliberately avoids the referenced characteristic; *specify*, which follows an explicit textual requirement rather than a reference; and *ignore*, which leaves the dimension unconstrained. We retain a sampled reference-dependent requirement only when the assigned reference contains sufficiently clear and discriminable evidence for the corresponding dimension.

Compositional query design. We construct a design matrix whose axes describe the user persona, target content, form and tone of the request, selected references, the reference-use specification $\mathcal{Z}_t$, and other relevant factors. Values are sampled across these axes and subsequently filtered to remove contradictory, implausible, or otherwise pathological combinations. An LLM then converts each valid combination into a fluent natural-language instruction. This procedure preserves linguistic naturalness while maintaining systematic control over task diversity. Additional generation details and examples are provided in Appendix B.2.

3.4 Verifiable Keypoint Construction

A query in RefCraft may admit many valid implementations, and its requirements can be fine-grained, subtle, and distributed across both language and visual references. Consequently, evaluating against a single canonical webpage or a holistic similarity score would be inappropriate. Inspired by rubric-based evaluation (Yu et al., 2026; Hashemi et al., 2024; Pathak et al., 2025), we instead decompose each query into a set of non-overlapping keypoints, each capturing one independently assessable requirement. We partition $\mathcal{K}_t = \mathcal{K}_t^{\text{prog}} \cup \mathcal{K}_t^{\text{vlm}}$ into disjoint tracks according to how each keypoint can be most reliably verified. *Program-verifiable keypoints* $\mathcal{K}_t^{\text{prog}}$ capture deterministic properties such as color distance, contrast, responsive behavior, DOM state changes, and robustness to abnormal content. The remaining keypoints require semantic or perceptual judgment and are assigned to the *VLM-assessed track* $\mathcal{K}_t^{\text{vlm}}$. We follow the principle that whenever reliable deterministic verification is possible, we favor the programmatic track to reduce reliance on VLM judgments, which can be overly lenient. We additionally designate essential user requirements and page-integrity constraints as critical items, ensuring that they receive greater emphasis in the final evaluation.

We apply a multi-stage validation procedure to ensure keypoint quality. A separate model first reviews the full set of keypoints for completeness and may add missing requirements. It then examines each keypoint independently to identify hallucinated, redundant, or unverifiable criteria. Finally, ten web development experts review and revise the resulting keypoints. This process produces a verified evaluation specification for every benchmark instance.

4 Evaluation Protocol

Given the benchmark instances constructed above, our evaluation protocol standardizes how models interact with the task, converts their unconstrained HTML outputs into structured evidence, and scores them against the corresponding keypoints.

4.1 Unified Agentic Environment

We evaluate each model as a coding agent, reflecting the increasingly common use of agentic harnesses in practical development workflows . To ensure comparability, all models operate through

the same open-source coding harness, `opencode` [3], with identical tools, workspace structure, and output requirements. For each task, the agent receives the user instruction, visual references, and an environment brief. It is given a dedicated working directory and may render, screenshot, and inspect intermediate implementations before revising them. The final result must be stored as a single self-contained HTML file, ensuring a uniform artifact format across models. We restrict access that could reveal benchmark internals or ground-truth evaluation criteria. Each working directory is isolated from the benchmark repository, and attempts to access the repository or evaluation keypoints are denied. Network access remains available to preserve a realistic coding environment, but searching for or downloading the original source of a reference webpage invalidates the run.

4.2 Evidence Collection

Generated HTML files vary substantially in structure and may exhibit visual and interactive behaviors that cannot be reliably evaluated from source code alone. Therefore, for each artifact $h_{m,t}$, we collect structured evidence $\mathcal{E}_{m,t} = (\mathcal{V}_{m,t}, \mathcal{M}_{m,t})$, where $\mathcal{V}_{m,t}$ contains captured views and $\mathcal{M}_{m,t}$ contains browser measurements. Specifically, for each benchmark instance, we derive a task-specific capture script specifying a sequence of actions, such as scrolling, hovering, clicking, and typing. The script is designed to expose the page states required to evaluate as many relevant keypoints as possible. We load the generated HTML in Playwright, execute the actions sequentially, and capture screenshots of the resulting states. In parallel, we apply a fixed suite of browser-based measurements to assess properties such as rendering integrity, responsive behavior, robustness to abnormal content, and whether interactive controls produce genuine state changes. Together, the screenshots and measurements provide downstream judges with targeted evidence rather than requiring them to infer page behavior from raw HTML, which would be poorly aligned with practical usage.

4.3 Dual-Track Judging and Score Aggregation

Our evaluation first produces keypoint-level scores from $\mathcal{E}_{m,t}$ using complementary programmatic and VLM judges. The programmatic judge compares browser measurements against deterministic keypoints and returns a binary verdict. For keypoints requiring semantic or perceptual judgment, the VLM judge receives the keypoint, relevant screenshots, and supporting measurements, and assigns a rating on a four-level scale from 0 to 3. Each keypoint is evaluated independently to reduce interference between unrelated requirements. Denote these judgments by $b_{t,k} \in \{0, 1\}$ and $r_{t,k} \in \{0, 1, 2, 3\}$, respectively, omitting the model index for clarity. Both tracks are normalized to the same percentage scale: $s_{t,k} = 100\, b_{t,k}$ for $k \in \mathcal{K}_t^{\text{prog}}$ and $s_{t,k} = 100\, r_{t,k}/3$ for $k \in \mathcal{K}_t^{\text{vlm}}$.

We aggregate these keypoint-level judgments into three complementary task-level quantities. The **base keypoint score** B_t is the mean of all normalized keypoint scores: $B_t = |\mathcal{K}_t|^{-1} \sum_{k \in \mathcal{K}_t} s_{t,k}$. The **core-function score** C_t captures the weakest essential functionality of the generated webpage. The core goals include renderability and task-specific essential interactions, and C_t is defined as the minimum completion rate across these goals. This metric reflects the intuition that even strong performance on peripheral requirements cannot compensate for a broken primary function, as such failures fundamentally undermine the user experience. The **reference-following score** R_t measures whether the generated webpage realizes the requested relationship to its visual references, which is the central capability evaluated by RefCraft. We compute R_t by first averaging the keypoint scores within each reference group and then averaging across all applicable groups. Because core functionality and reference adherence are both essential to user-perceived quality, C_t and R_t act as gates on the aggregate keypoint score. The final task score therefore takes the minimum over the applicable quantities rather than naively averaging them:

$$S_t = \begin{cases} \min\{B_t, C_t, R_t\}, & \text{if } R_t \text{ is defined,} \\ \min\{B_t, C_t\}, & \text{otherwise,} \end{cases} \qquad \bar{S}_m = \frac{1}{|\mathcal{T}_m|} \sum_{t \in \mathcal{T}_m} S_t,$$

where $\mathcal{T}_m$ denotes the set of tasks evaluated for model m. Additional details on judging and score aggregation are provided in Appendix B.5.

[3] https://github.com/anomalyco/opencode

Table 1: Results on the 300-task RefCraft benchmark. We report Overall and its three component scores: keypoint completion (Keypts., B), core functionality (Core, C), and reference following (Ref., R). Rollout statistics report mean agent turns, total tokens (Total Tok.), and tool calls per task. Bold and underlined values indicate the best and second-best entries in each column, respectively (higher scores and lower resource counts).

	Evaluation				Rollout Statistics		
Model	**Overall↑**	**Keypts. B ↑**	**Core C ↑**	**Ref. R ↑**	**Turns**	**Total Tok.†**	**Calls**
GPT-6 Astra	**52.12**	**75.68**	**94.89**	57.93	21.7	1.40 M	30.1
Qwen3.8-Max	<u>48.95</u>	<u>73.86</u>	<u>85.56</u>	62.61	**9.8**	**0.68 M**	**11.3**
Grok 4.6	48.83	72.96	81.17	**66.25**	21.2	1.28 M	41.2
Kimi K3	47.55	72.74	85.00	61.33	25.4	2.09 M	31.3
Claude Fable 5	44.61	72.05	77.83	61.44	<u>15.2</u>	1.10 M	<u>16.9</u>
GLM-5.3	42.60	71.23	83.56	55.77	38.3	2.99 M	42.2
DeepSeek-V4-Pro	37.09	71.16	71.72	53.46	25.3	1.12 M	25.6
Seed2.1 Pro	32.06	69.00	56.56	58.67	22.7	<u>1.01 M</u>	21.8
Gemini 3.1 Pro	19.01	61.45	33.44	<u>65.00</u>	26.7	2.34 M	25.9

† Token counts are taken from API usage reports and sum input, cache-read, cache-write, output, and reasoning tokens. M denotes millions.

5 Experiments

5.1 Experimental Setup

We evaluate nine frontier models—GPT-6 Astra (OpenAI, 2026d), Claude Fable 5 (Anthropic, 2026a), Qwen3.8-Max (Qwen Team, 2026b), Grok 4.6 (xAI, 2026), Kimi K3 (Kimi et al., 2026), GLM-5.3 (Z.ai, 2026), DeepSeek-V4-Pro (Xu et al., 2026), Seed2.1 Pro (ByteDance Seed Team, 2026b), and Gemini 3.1 Pro (Google DeepMind, 2026)—as coding agents on RefCraft. All models are evaluated using the same `OpenCode` v1.18.25 harness, with high reasoning effort and a maximum of 100 turns. Keypoints are generated by Claude Opus 5 (Anthropic, 2026f) and verified by GPT-5.6 Sol (OpenAI, 2026c). We use Gemini 3.1 Pro as the VLM judge. We report the overall score S together with its three constituent metrics. Further details are provided in Appendix B.

5.2 Main Results

Tab. 1 presents the overall and component scores of the evaluated frontier models, together with efficiency statistics such as token usage and the number of interaction turns. From these results, we could make the following observations:

Visual-reference-conditioned generation remains far from solved. As shown in Tab. 1, GPT-6 Astra achieves the highest overall score of 52.12 out of 100, most models cluster around 40, and Gemini 3.1 Pro scores below 20. These results leave substantial room for improvement. Moreover, the relatively low overall scores contrast with the higher and more concentrated keypoint scores of 61.45–75.68, indicating that models can satisfy many individual requirements yet struggle to produce artifacts that simultaneously preserve essential functionality and realize the user-specified relationship to the references. This gap captures the central challenge posed by RefCraft: a visually plausible webpage cannot be considered successful if it uses references inconsistently with the user's instructions, reflecting how users assess artifacts in real-world settings.

Reference use and functional execution constitute distinct bottlenecks. For seven of the nine models, the reference-following score R is the lowest component, identifying reference utilization as the dominant bottleneck. However, reference use alone does not determine end-to-end performance: Gemini 3.1 Pro achieves an R of 65.00 but ranks ninth overall because its core-function score is only 33.44, whereas GPT-6 Astra leads with the strongest core-function score of 94.89 despite a lower R. These contrasting failure regimes demonstrate that reliable implementation and appropriate reference utilization capture complementary capabilities, motivating their joint consideration in RefCraft.

More budget does not necessarily yield more effective performance. Performance is not monotonic in the number of turns, total reported tokens, or tool calls. Qwen3.8-Max outperforms Grok 4.6 while using substantially fewer total tokens (0.68M versus 1.28M), turns, and tool calls. GLM-5.3

Table 2: Benefits of references and revision.

Setting	GPT-6 Astra	DeepSeek-V4-Pro
(a) Reference ablation		
Without references	17.3	11.1
With references	52.1	37.1
Gain	**+34.8**	**+26.0**
(b) Inspection and revision		
Initial	47.4	30.7
Final	52.1	37.1
Gain	**+4.7**	**+6.4**

Table 3: Human evaluation agreement.

Category	Human–human Krippendorff α	Evaluator vs. expert median Weighted Cohen κ
(a) Keypoint-level agreement		
Visual attributes	0.79	0.72
Visual style	0.65	0.47
Interaction	0.70	0.55
All keypoints	**0.72**	**0.64**
(b) Task-level Pearson r		
Evaluator vs. mean expert score		**0.74**

uses the most turns, total tokens (2.99M), and tool calls, yet obtains a substantially lower overall score than the leading models. We further present a cost–efficiency Pareto curve in Fig. 1, which highlights the distinct efficiency profiles of different models.

5.3 Further Analysis

5.3.1 Fine-Grained Capability Analysis

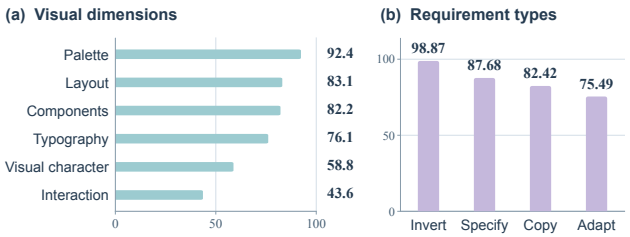


Figure 3: Fine-grained performance by (a) visual dimension and (b) requirement type, averaged equally across models.

We further analyze model performance across fine-grained categories, including visual dimensions and requirement types. Additional results are provided in Appendix A.5.

Fig. 3(a) presents performance across the six visual dimensions defined in § 3. Using an equal-weight average across models, palette (92.4), layout (83.1), and components (82.2) score higher than typography (76.1) and overall visual character (58.8). Interaction achieves the lowest score (43.6) and ranks last for nearly all evaluated systems. These results indicate that agents transfer concrete, locally observable attributes more reliably than properties requiring coordination across multiple design choices or interactive states. Disaggregating keypoints by requirement type reveals differences in difficulty (Fig. 3 (b)). Models perform best on *invert* (98.87) and *specify* (87.68), followed by *copy* (82.42), whereas *adapt* receives the lowest score (75.49). This suggests that selectively transforming a referenced attribute to fit a new context is more challenging than directly reproducing, explicitly specifying, or avoiding it, despite being central to practical reference-guided development.

5.3.2 Reference Utilization and Visual Inspection Capabilities

As discussed in § 1, effectively incorporating visual references and refining intermediate outputs through visual inspection are important capabilities in agentic generation. We assess these capabilities via our RefCraft. **(1) Reference ablation.** We remove all visual references while leaving the original user query unchanged. The agent is explicitly informed that the references mentioned in the query are unavailable and is instructed to satisfy all requirements using its prior knowledge to make reasonable choices. **(2) Pre- and post-inspection comparison.** We evaluate the artifact produced in the first generation round, before the agent receives a rendered view of its output, and compare it with the final artifact. The results are reported in Tab. 2. Removing the visual references substantially reduces performance for both models. GPT-6 Astra drops from 52.1 to 17.3 points in overall score, while DeepSeek-V4-Pro drops from 37.1 to 11.1. These results indicate that effectively incorporating the supplied visual references is a crucial capability and that models cannot recover the reference-dependent requirements from the textual query and their priors alone. We also observe that performance improves from the initial to the final artifacts when models have additional opportunities to inspect and revise their outputs. Although this comparison does not rigorously isolate the causal

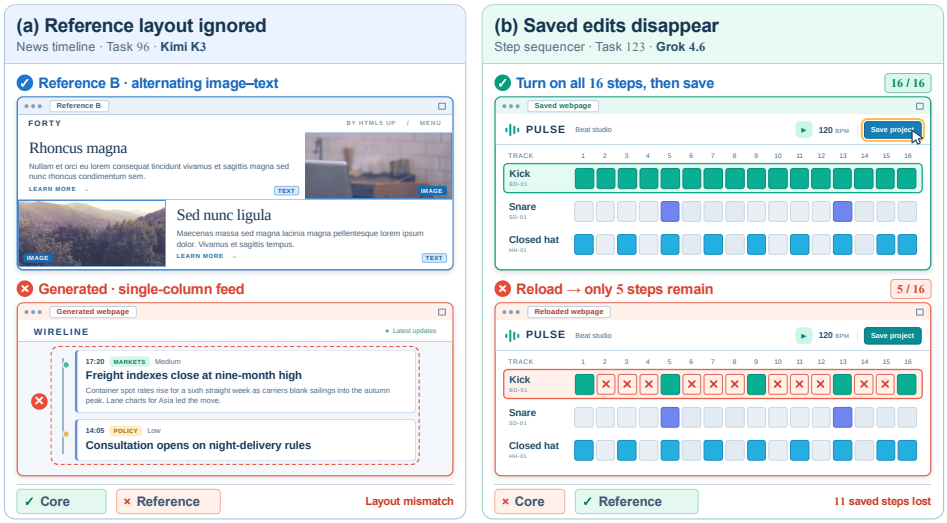


Figure 4: Two failure cases identified by RefCraft: one in reference following and the other in core functionality.

contribution of visual inspection, since the final artifacts also benefit from additional tool calls, the consistent improvements suggest that the complete inspection-and-revision process is important.

5.3.3 Evaluation Reliability and Qualitative Analysis

To assess the reliability of our evaluation protocol, we sample 100 model–task artifacts and recruit ten web-development experts to evaluate each artifact independently. We aggregate the expert judgments using the median label for each keypoint and the mean score for each task. As shown in Tab. 3, the evaluator achieves substantial agreement with the expert consensus, both overall and across evaluation aspects, with a weighted κ of 0.64 across all keypoints. More subjective criteria, such as overall visual character, exhibit greater human–human and evaluator–human disagreement, as expected. The task-level scores also correlate strongly with the mean expert scores (Pearson $r = 0.747$), indicating that this alignment extends to aggregate assessments of task performance. We further compare our overall model ranking with the Image-to-Webdev leaderboard from Arena.AI, a popular platform for human evaluation of frontier models[4]. The two rankings achieve a Spearman correlation of $\rho = 0.89$, suggesting broad consistency between RefCraft and real-world user needs and preferences.

Fig. 4 presents two qualitative failure cases detected by RefCraft. In Task 96, Kimi K3 produces a polished news-timeline page that satisfies many keypoints and all core goals. Yet it adopts a centered, single-column feed rather than the requested full-width, two-column image–text layout, thereby failing the reference requirement. In Task 123, Grok 4.6 produces a step sequencer that follows the visual references, but its save function fails to retain edits, although edit persistence is explicitly requested in the query and can be implemented in JavaScript. Together, these cases illustrate how RefCraft identifies both unmet reference requirements and broken core functionality.

6 Conclusion

In this paper, we introduced RefCraft, a benchmark for studying reference-based webpage development and, more broadly, visual-reference-conditioned agentic generation in frontier models. RefCraft comprises 300 realistic web-development tasks in an inspect-and-iterate setting. We decompose each query into independently verifiable keypoints and evaluate the generated artifacts using complementary programmatic and VLM judges. Our evaluation of nine frontier (M)LLMs shows that this capability remains far from solved: the strongest system achieves an overall score of only 52.12. We hope that RefCraft will provide a valuable testbed for this crucial yet underexplored capability and stimulate further progress in visual-reference-conditioned agentic generation.

[4] https://arena.ai/leaderboard/code/image-to-webdev, accessed September 24, 2026.

AI USE STATEMENT

We used generative AI models in both benchmark construction and evaluation, including synthesizing reference webpages, profiling visual evidence, constructing and reviewing tasks and keypoints, and performing VLM-based evaluation (§ B). AI assistants also supported manuscript preparation, including routine tool development and language polishing. The authors reviewed all AI-assisted content and take full responsibility for the final text, experimental claims, and research artifacts.

REPRODUCIBILITY STATEMENT

Task construction, scoring, and implementation details are documented in § 3, § 4, and § B, including prompts, browser settings, and execution budgets. Upon publication, we plan to publicly release RefCraft, including task inputs, evaluation scripts, previously generated HTML artifacts, run records, and item-level judgments.

REFERENCES

Anthropic. System Card: Claude Opus 4.5. Technical report, Anthropic, November 2025. URL https://www.anthropic.com/claude-opus-4-5-system-card.

Anthropic. Claude Fable 5 & Claude Mythos 5 System Card. Technical report, June 2026a. URL https://www-cdn.anthropic.com/d00db56fa754a1b115b6dd7cb2e3c342ee809620.pdf.

Anthropic. Claude Fable 5.1 and Claude Mythos 5.1, September 2026b. URL https://www.anthropic.com/claude-fable-and-mythos-5-1. Accessed: 2026-09-12.

Anthropic. System Card: Claude Opus 4.6. Technical report, Anthropic, February 2026c. URL https://www.anthropic.com/claude-opus-4-6-system-card.

Anthropic. System Card: Claude Opus 4.8. Technical report, Anthropic, May 2026d. URL https://www.anthropic.com/claude-opus-4-8-system-card.

Anthropic. System Card: Claude Sonnet 4.6. Technical report, Anthropic, February 2026e. URL https://www.anthropic.com/claude-sonnet-4-6-system-card.

Anthropic. Introducing Claude Opus 5. Technical report, Anthropic, 2026f. URL https://www.anthropic.com/news/claude-opus-5.

Jianhong Bai, Tianyu He, Yuchi Wang, Junliang Guo, Haoji Hu, Zuozhu Liu, and Jiang Bian. Uniedit: A unified tuning-free framework for video motion and appearance editing. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 10171–10180, 2025.

ByteDance Seed Team. Seed2.0 Model Card: Towards Intelligence Frontier for Real-World Complexity. Technical report, ByteDance, February 2026a. URL https://lf3-static.bytednsdoc.com/obj/eden-cn/lapzild-tss/ljhwZthlaukjlkulzlp/seed2/0214/Seed2.0%20Model%20Card.pdf.

ByteDance Seed Team. Seed2.1 Model Card. Technical report, June 2026b. URL https://lf3-static.bytednsdoc.com/obj/eden-cn/lapzild-tss/ljhwZthlaukjlkulzlp/seed2.1/Seed2_1_Model_Card.pdf. Covers Seed2.1 Pro and Seed2.1 Turbo.

Liang Chen, Yichi Zhang, Shuhuai Ren, Haozhe Zhao, Zefan Cai, Yuchi Wang, Peiyi Wang, Xiangdi Meng, Tianyu Liu, and Baobao Chang. Pca-bench: Evaluating multimodal large language models in perception-cognition-action chain. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 1086–1104, 2024.

Yang Chen, Minghao Liu, Yufan Shen, Yunwen Li, Tianyuan Huang, Xinyu Fang, Tianyu Zheng, Wenxuan Huang, Cheng Yang, Licheng Wen, et al. Iwr-bench: Can lvlms reconstruct interactive webpage from a user interaction video? In *International Conference on Learning Representations*, volume 2026, pp. 52158–52189, 2026.

GLM-5 Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, et al. GLM-5: from Vibe Coding to Agentic Engineering. *arXiv preprint arXiv:2602.15763*, 2026. URL https://arxiv.org/abs/2602.15763.

Google DeepMind. Gemini 3.1 Pro Model Card. Technical report, February 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/. Covers the gemini-3.1-pro-preview API model.

Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. Llm-rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 13806–13834, 2024.

Vanita Jain, Piyush Agrawal, Subham Banga, Rishabh Kapoor, and Shashwat Gulyani. Sketch2code: transformation of sketches to ui in real-time using deep neural network. *arXiv preprint arXiv:1910.08930*, 2019.

Team Kimi, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Xinping Lei, Xinyu Che, Junqi Xiong, Chenchen Zhang, Yukai Huang, Chenyu Zhou, Haoyang Huang, Minghao Liu, Letian Zhu, Hongyi Ye, et al. Webcompass: Towards multimodal web coding evaluation for code language models. *arXiv preprint arXiv:2604.18224*, 2026.

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*, 38, 2026.

MiniMax. MiniMax M2.7: Early Echoes of Self-Evolution, March 2026. URL https://www.minimax.io/news/minimax-m27-en.

OpenAI. Introducing GPT-5.2, December 2025. URL https://openai.com/index/introducing-gpt-5-2/.

OpenAI. Introducing GPT-5.4, March 2026a. URL https://openai.com/index/introducing-gpt-5-4/.

OpenAI. GPT-5.5 Model, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5.5. Accessed: 2026-09-12.

OpenAI. GPT-5.6 Sol Model, 2026c. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol.

OpenAI. GPT-6 Astra Model, 2026d. URL https://developers.openai.com/api/docs/models/gpt-6-astra. Accessed: 2026-09-12.

Aditya Pathak, Rachit Gandhi, Vaibhav Uttam, Arnav Ramamoorthy, Pratyush Ghosh, Aaryan Raj Jindal, Shreyash Verma, Aditya Mittal, Aashna Ased, Chirag Khatri, et al. Rubric is all you need: Improving llm-based code evaluation with question-specific rubrics. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V. 1*, pp. 181–195, 2025.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Qwen Team. Qwen3.6-Plus: Towards Real World Agents, April 2026a. URL https://qwen.ai/blog?id=qwen3.6.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL https://qwen.ai/blog?id=qwen3.8.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22500–22510. IEEE, 2023.

Siddhant Saxena, Nilesh Trivedi, and Vinayaka Jyothi. Swe-webdevbench: Evaluating coding agent application platforms as virtual software agencies. *arXiv preprint arXiv:2605.04637*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Team Seedance, De Chen, Liyang Chen, Xin Chen, Ying Chen, Zhuo Chen, Zhuowei Chen, Feng Cheng, Tianheng Cheng, Yufeng Cheng, et al. Seedance 2.0: Advancing video generation for world complexity. *arXiv preprint arXiv:2604.14148*, 2026.

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.

Haoyu Sun, Huichen Will Wang, Jiawei Gu, Linjie Li, and Yu Cheng. Fullfront: Benchmarking mllms across the full front-end engineering workflow. *arXiv preprint arXiv:2505.17399*, 2025.

Hung Tran, Langston Nashold, Rayan Krishnan, Antoine Bigeard, and Alex Gu. Vibe code bench: Evaluating ai models on end-to-end web application development. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pp. 514–536, 2026.

Kuang-Da Wang, Zhao Wang, Wei-Yao Wang, Yotaro Shimose, and Shingo Takamatsu. Webgen-v bench: Structured representation for enhancing visual design in llm-based web generation and evaluation. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 9927–9938, 2026.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025a.

Yuchi Wang, Junliang Guo, Jianhong Bai, Runyi Yu, Tianyu He, Xu Tan, Xu Sun, and Jiang Bian. Instructavatar: Text-guided emotion and motion control for avatar generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 8132–8140, 2025b.

Chenyuan Wu, Pengfei Zheng, Ruiran Yan, Shitao Xiao, Xin Luo, Yueze Wang, Wanli Li, Xiyan Jiang, Yexin Liu, Junjie Zhou, et al. Omnigen2: Exploration to advanced multimodal generation. *arXiv e-prints*, pp. arXiv–2506, 2025.

xAI. Introducing Grok 4.6, August 2026. URL https://x.ai/news/grok-4-6.

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.

Shitao Xiao, Yueze Wang, Junjie Zhou, Huaying Yuan, Xingrun Xing, Ruiran Yan, Chaofan Li, Shuting Wang, Tiejun Huang, and Zheng Liu. Omnigen: Unified image generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13294–13304. IEEE, 2025c.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Kai Xu, YiWei Mao, XinYi Guan, and ZiLong Feng. Web-bench: A llm code benchmark based on web standards and frameworks. *arXiv preprint arXiv:2505.07473*, 2025.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Kaiyu Yang, Yuyan Bu, Jing Yi, Yuchi Wang, Biyu Zhou, Juntao Dai, Songlin Hu, and Yaodong Yang. When lower privileges suffice: Investigating over-privileged tool selection in llm agents. *ArXiv*, abs/2606.20023, 2026. URL https://api.semanticscholar.org/CorpusID:289626701.

Zhen Yang, Wenyi Hong, Mingde Xu, Xinyue Fan, Weihan Wang, Jiale Cheng, Xiaotao Gu, and Jie Tang. Ui2codeˆ n: Ui-to-code generation as interactive visual optimization. *arXiv preprint arXiv:2511.08195*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Hu Ye, Jun Zhang, Sibo Liu, Xiao Han, and Wei Yang. Ip-adapter: Text compatible image prompt adapter for text-to-image diffusion models. *arXiv preprint arXiv:2308.06721*, 2023.

Weihao Yu, Rongyao Fang, Yuxuan Cai, Linjiang Huang, Yuhuan Yang, Xianwei Zhuang, Junyang Lin, Yixuan Yuan, and Shuai Bai. Specv: Specification verification for robust unified multimodal evaluation. In *European Conference on Computer Vision*, pp. 59–75. Springer, 2026.

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-5.3: Frontier Coding with Emergent Cyber Capabilities, 2026. URL https://z.ai/blog/glm-5.3.

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.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3813–3824. IEEE, 2023.

Hongda Zhu, Yiwen Zhang, Bing Zhao, Jingzhe Ding, Siyao Liu, Tong Liu, Dandan Wang, Yanan Liu, and Zhaojian Li. Frontendbench: A benchmark for evaluating llms on front-end development via automatic evaluation. *arXiv preprint arXiv:2506.13832*, 2025.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.

APPENDIX OVERVIEW

We organize the appendix as follows:

- Appendix A presents additional experiments and analyses, including:
  - benchmark statistics (Appendix A.1)
  - pairwise model comparisons (Appendix A.2)
  - judge sensitivity and stability (Appendix A.3)
  - evaluation protocol ablations (Appendix A.4)
  - complete capability breakdowns (Appendix A.5)
  - cross-harness sensitivity (Appendix A.6)
  - a worked example of RefCraft (Appendix A.7)
- Appendix B provides additional methodological details, including:
  - reference collection and visual evidence (Appendix B.1)
  - controlled task construction (Appendix B.2)
  - keypoint construction and quality assurance (Appendix B.3)
  - standardized agentic execution (Appendix B.4)
  - evidence collection and dual-track judging (Appendix B.5)

A ADDITIONAL EXPERIMENTS

The main experiments establish aggregate model performance and identify challenges in reference utilization and functional execution. Here, we examine how these results vary across individual tasks, and how their interpretation depends on the evaluation and execution setup. These analyses complement the main comparison by distinguishing performance differences from variation introduced by the judge, scoring protocol, or coding harness.

A.1 BENCHMARK STATISTICS

We summarize the composition of RefCraft in Fig. 5. The final 300 tasks span eight application domains, 40 subdomains, and 186 distinct webpage types. Each task uses one to five reference webpages, with three- and four-reference tasks accounting for 78 and 72 instances, respectively. The benchmark also varies the intended degree of reference use, from loose guidance to near-copying, as determined by requirement types such as *adapt* and *copy*, and includes 151 English and 149 Chinese requests. Together, these distributions capture variation in both what agents are asked to build and how they should use the supplied references. The 744-page reference pool is counted separately because each source page may support multiple tasks.

During evaluation, each keypoint is instantiated as a single evaluation record. This process yields 12,682 records, comprising 10,542 VLM-assessed criteria, 1,540 programmatic probes, and 600 renderability records. The first two categories form the 12,082-item score-bearing set, whereas the renderability records support auditing and core evaluation without contributing additional scoring items. Within the score-bearing set, 3,106 criteria assess directive-related requirements; Fig. 5(d) presents their breakdown.

A.2 PAIRWISE MODEL COMPARISONS

We provide detailed task-paired comparisons among the nine evaluated models in Tab. 4. For each model pair, we compute the mean difference in overall score across the same tasks and report the corresponding proportions of wins, ties, and losses. The head-to-head results are broadly consistent with the aggregate ranking, while showing that a higher-ranked model does not uniformly outperform its counterpart. In particular, comparisons between closely ranked systems contain substantial proportions of both ties and losses. These paired results clarify how a leaderboard advantage is distributed across tasks, rather than treating a difference in average score as evidence of uniformly superior performance.

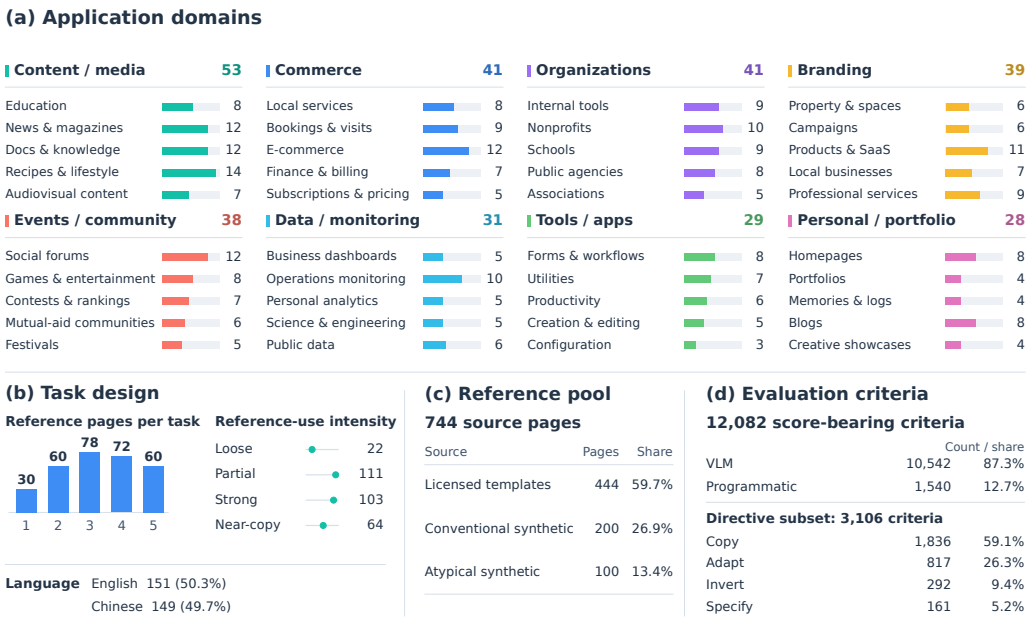


Figure 5: Composition and diversity of RefCraft.

A.3 Judge Sensitivity and Stability

In our evaluation, we use Gemini 3.1 Pro as the VLM judge. A natural question is whether the broad model comparison depends on the choice of automatic evaluator. We examine this sensitivity by re-scoring artifacts from the nine evaluated systems on a fixed 32-task sample using Gemini 3.1 Pro, GPT-6 Astra, and Claude Opus 5. Each judge receives identical tasks, keypoints, reference mappings, rendered views, browser measurements, and interaction evidence. As shown in Fig. 6, the three judges produce similar model-level score patterns, suggesting that the broad comparison on this sample is not specific to a single evaluator.

We further examine the stability of the same judge across repeated evaluations. Specifically, we run Gemini 3.1 Pro three times on the same sample with identical inputs. Pairwise correlations between repeated scores reach 0.92, indicating that the overall score pattern is reproducible across runs.

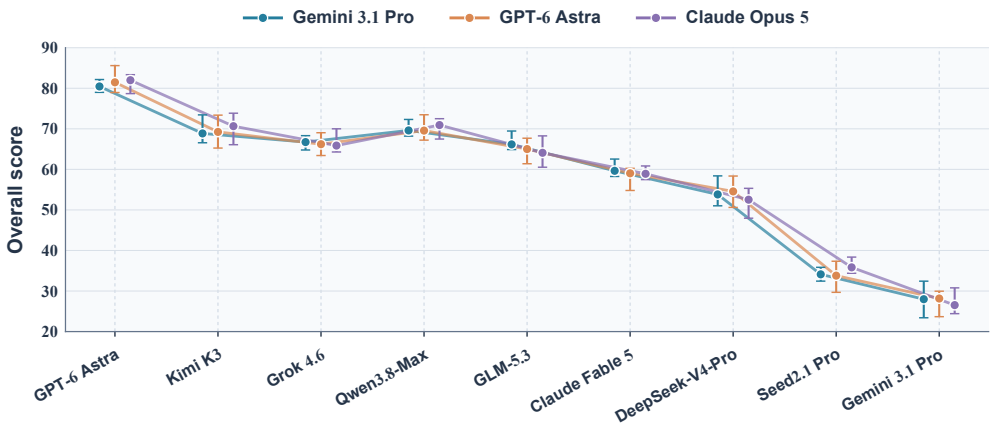


Figure 6: Judge sensitivity on the fixed 32-task sample. Colored points and connecting lines show scores assigned by Gemini 3.1 Pro, GPT-6 Astra, and Claude Opus 5. Whiskers indicate illustrative uncertainty intervals. Relative scoring patterns are broadly consistent across the three models.

Table 4: Paired comparisons among the nine main-table models. Δ is Model A minus Model B.

Model A	Model B	Δ	A win (%)	Tie (%)	B win (%)
GPT-6 Astra	Qwen3.8-Max	3.17	38.3	24.7	37.0
GPT-6 Astra	Grok 4.6	3.29	42.7	21.7	35.7
GPT-6 Astra	Kimi K3	4.57	44.0	20.7	35.3
GPT-6 Astra	Claude Fable 5	7.51	46.7	20.7	32.7
GPT-6 Astra	GLM-5.3	9.52	47.3	23.7	29.0
GPT-6 Astra	DeepSeek-V4-Pro	15.03	51.0	25.3	23.7
GPT-6 Astra	Seed2.1 Pro	20.06	61.3	22.0	16.7
GPT-6 Astra	Gemini 3.1 Pro	33.11	71.0	20.3	8.7
Qwen3.8-Max	Grok 4.6	0.12	37.0	26.0	37.0
Qwen3.8-Max	Kimi K3	1.40	41.3	24.0	34.7
Qwen3.8-Max	Claude Fable 5	4.34	41.0	25.0	34.0
Qwen3.8-Max	GLM-5.3	6.35	44.7	28.0	27.3
Qwen3.8-Max	DeepSeek-V4-Pro	11.86	48.0	30.3	21.7
Qwen3.8-Max	Seed2.1 Pro	16.89	55.7	26.0	18.3
Qwen3.8-Max	Gemini 3.1 Pro	29.94	68.0	23.7	8.3
Grok 4.6	Kimi K3	1.28	42.3	24.0	33.7
Grok 4.6	Claude Fable 5	4.21	39.7	27.3	33.0
Grok 4.6	GLM-5.3	6.23	43.3	23.3	33.3
Grok 4.6	DeepSeek-V4-Pro	11.74	43.7	32.7	23.7
Grok 4.6	Seed2.1 Pro	16.77	54.0	28.0	18.0
Grok 4.6	Gemini 3.1 Pro	29.82	67.0	26.7	6.3
Kimi K3	Claude Fable 5	2.94	37.3	26.0	36.7
Kimi K3	GLM-5.3	4.95	41.3	26.3	32.3
Kimi K3	DeepSeek-V4-Pro	10.46	44.3	32.7	23.0
Kimi K3	Seed2.1 Pro	15.49	53.0	26.0	21.0
Kimi K3	Gemini 3.1 Pro	28.54	67.7	22.7	9.7
Claude Fable 5	GLM-5.3	2.02	40.3	26.7	33.0
Claude Fable 5	DeepSeek-V4-Pro	7.52	43.3	33.0	23.7
Claude Fable 5	Seed2.1 Pro	12.56	49.3	31.0	19.7
Claude Fable 5	Gemini 3.1 Pro	25.60	61.3	29.0	9.7
GLM-5.3	DeepSeek-V4-Pro	5.51	37.3	35.7	27.0
GLM-5.3	Seed2.1 Pro	10.54	47.7	29.0	23.3
GLM-5.3	Gemini 3.1 Pro	23.59	60.0	27.0	13.0
DeepSeek-V4-Pro	Seed2.1 Pro	5.03	37.7	39.0	23.3
DeepSeek-V4-Pro	Gemini 3.1 Pro	18.08	51.7	35.0	13.3
Seed2.1 Pro	Gemini 3.1 Pro	13.05	40.7	46.0	13.3

A.4 Evaluation Protocol Ablations

The protocol in § 4 combines requirement-level assessment with visual and executable evidence. To examine the role of these design choices, we compare the complete protocol with two plausible alternatives on the same generated artifacts. *Holistic evaluation* replaces keypoint decomposition and $B/C/R$ aggregation with a direct overall rating, testing how the assessment changes when requirements are no longer scored and aggregated separately. *Screenshot-only evaluation* retains the rendered images but removes browser measurements and action traces, testing what can be established from visual evidence alone.

As shown in Fig. 7, holistic scores exceed complete-protocol scores by 42.3 points. Because holistic evaluation changes both the rating granularity and the aggregation rule, this gap alone cannot establish which protocol better captures artifact quality. Nevertheless, the large increase suggests that holistic evaluation may overestimate model capabilities by overlooking failures exposed by the complete protocol. Removing executable evidence has a clearer effect on evaluation coverage. Unresolved core cases increase from 26 to 437 out of 918, while unresolved keypoints increase from 15 to 1,987 out of 7,608. These results indicate that visual evidence alone is often insufficient to determine whether requirements are satisfied, making screenshot-only evaluation incomplete and unreliable for assessing overall artifact behavior.

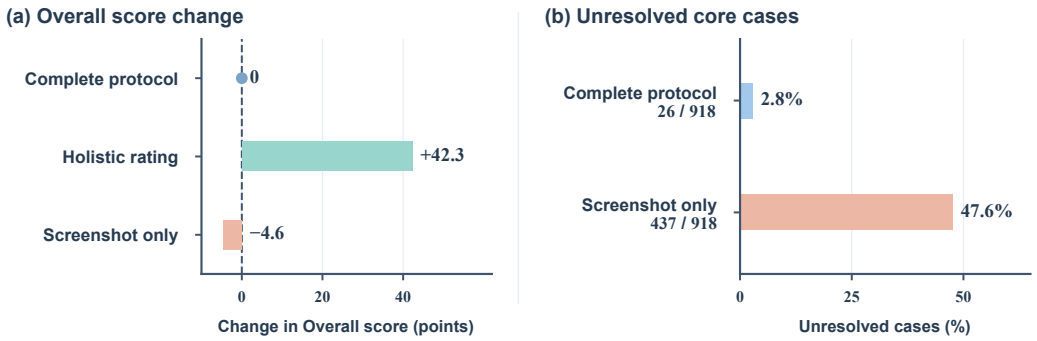


Figure 7: Protocol sensitivity and evidence coverage. (a) Changes in Overall score relative to the complete protocol: $+42.3$ under holistic rating and -4.6 under screenshot-only evaluation. (b) Observed rates of unresolved core cases across matched artifacts.

A.5 Complete Capability Breakdowns

The fine-grained analysis in § 5.3.1 focuses on visual dimensions and requirement types. Tab. 6 and 5 complement this analysis by examining application domains and evidence sources across the nine evaluated models. The domain breakdown reports task-level overall scores, capturing end-to-end performance within each task group. The source breakdown instead reports reference-use keypoint scores, obtained by first aggregating the relevant keypoints within each task and then averaging across tasks. Across domains, equal-model averages are highest for personal and portfolio (44.7), organizations (44.4), and commerce (44.1), and lowest for data and monitoring (33.7) and events and community (37.1). Tab. 5 shows that models score higher on text-specified requirements than on requirements associated with atypical synthetic, licensed/template, or conventional synthetic references. This gap suggests that explicit textual constraints are easier to follow than properties that must be inferred from visual references.

Table 5: Reference-related keypoint performance by evidence source.

Model	Licensed/ template	Atypical synthetic	Conventional synthetic	Text specification
GPT-6 Astra	82.7	78.8	80.7	91.6
Qwen3.8-Max	81.9	80.0	88.5	90.9
Grok 4.6	83.3	82.5	87.4	91.1
Kimi K3	81.6	79.3	81.7	85.5
Claude Fable 5	81.7	78.5	88.0	87.4
GLM-5.3	80.5	75.7	80.8	88.1
DeepSeek-V4-Pro	79.3	76.0	80.3	86.1
Seed2.1 Pro	80.0	76.8	81.9	81.8
Gemini 3.1 Pro	80.9	81.7	83.7	86.6

Table 6: Task-level Overall score across the eight high-level application domains.

Model	Content/ media	Commerce	Organizations	Branding	Events/ community	Data/ monitoring	Tools/ apps	Personal/ portfolio
GPT-6 Astra	50.2	57.6	53.9	54.4	46.8	46.0	54.4	53.6
Qwen3.8-Max	49.2	50.5	50.2	49.4	42.2	50.0	49.4	51.2
Grok 4.6	46.0	52.0	55.6	49.3	49.6	34.7	49.6	52.6
Kimi K3	43.9	49.4	53.5	46.8	39.5	44.5	51.2	54.7
Claude Fable 5	42.6	50.6	50.7	43.8	43.5	35.8	46.9	40.9
GLM-5.3	45.9	40.4	45.8	39.9	38.2	35.7	48.2	46.7
DeepSeek-V4-Pro	40.3	38.7	37.7	38.2	32.4	25.0	37.4	45.6
Seed2.1 Pro	33.9	37.9	32.2	38.8	22.0	20.2	33.0	36.4
Gemini 3.1 Pro	16.1	19.8	20.4	20.3	19.9	11.3	24.7	21.0

A.6 Cross-Harness Sensitivity

The shared `OpenCode` configuration ensures a consistent main comparison across models, but real-world agent systems use diverse harnesses. To test sensitivity to the choice of harness, we evaluate GPT-6 Astra and Kimi K3 on a fixed 32-task sample using two popular alternatives: Claude Code[5] and Codex[6]. We also include the released OpenCode artifacts for the same 32 tasks, re-scored using the same evaluation protocol, as the main-harness baseline.

For the Claude Code and Codex reruns, we match the task text, ordered references, initial workspaces, and nominal resource limits, while retaining each harness's native system prompts, tools, image preprocessing, and context-management mechanisms. Tab. 7 summarizes the overall scores for each model–harness combination. Scores vary modestly across harnesses. GPT-6 Astra ranges from 49.2 to 53.0, a spread of 2.8 points, while Kimi K3 ranges from 43.1 to 49.4, a spread of 1.1 points. GPT-6 Astra outperforms Kimi K3 under all three harnesses, preserving their relative ordering. These results suggest that the harness affects absolute scores, while overall performance and model ordering remain relatively stable on this sample.

Table 7: Cross-harness sensitivity on the fixed 32-task sample. We report Overall scores across three harnesses.

Model	Harness	Overall score
GPT-6 Astra	OpenCode (released)	49.2
GPT-6 Astra	Claude Code	51.3
GPT-6 Astra	Codex	53.0
Kimi K3	OpenCode (released)	48.2
Kimi K3	Claude Code	49.4
Kimi K3	Codex	43.1

A.7 A Worked Example of RefCraft

Fig. 9 traces how GPT-6 Astra, running in OpenCode, completes Task 163: a marine repair quote builder. The model first identifies which properties to borrow from each reference and which to avoid, combining bold typography and light service cards while rejecting the dark reference palette. It then builds a self-contained `index.html`, inspects rendered views, and revises the tablet heading and booking controls. Functional checks verify that repair selections update the quote and that editing a confirmed plan reopens the booking form; the final page also shows no horizontal overflow across 13 tested widths. This example illustrates how reference interpretation, visual refinement, and interaction testing work together to turn a user request into a functional webpage.

B Additional Implementation Details

§ 3 and § 4 describe the design of RefCraft and its evaluation protocol. Here, we explain how these designs are implemented. We follow the same progression as the main text: reference webpages are first converted into visual evidence, which supports the construction of user requests and their evaluation keypoints. Once these task components are fixed, an agent uses the request and reference images to develop a webpage, while a separate evaluator assesses the submitted artifact against the hidden keypoints. The details below apply to the RefCraft benchmark and clarify how information passes between these stages.

B.1 Reference Collection and Visual Evidence

We begin by preparing a reusable collection of webpages from which task-specific references can be assembled. Following the three-source design in § 3.2, this collection contains 744 pages: 444 openly licensed template-derived pages, 200 conventionally styled synthetic pages, and 100 atypical synthetic pages. These counts describe the available source pool rather than the benchmark instances.

[5] https://github.com/anthropics/claude-code
[6] https://github.com/openai/codex

A task draws visual references from one to five source pages, and the same source page may support different requests that ask agents to use its properties in different ways.

Template sources and licensing. For the template-derived portion, we collect openly licensed pages from collections including HTML5 UP, Colorlib, Start Bootstrap, Creative Tim, AdminLTE, Themefisher, and CodeWithSadee. The retained copies comprise 367 MIT-licensed pages, 76 under CC BY 3.0, and one under Apache-2.0. For each page, we record its source URL, author, license, and design-family identifier, preserving the connection between the benchmark reference and its original source. We also package the page's styles, scripts, and image assets into a self-contained HTML snapshot. This snapshot provides a stable input to the capture process: the browser renders the stored page rather than relying on the continued availability or current appearance of the live website.

Synthetic reference generation. The synthetic portion complements these existing templates by extending the range of designs available for task construction. Across the 300 synthetic pages, we use GPT-5.2 and GPT-5.4 (OpenAI, 2025; 2026a), Claude Opus 4.5, Claude Opus 4.6, Claude Opus 4.8, and Claude Sonnet 4.6 (Anthropic, 2025; 2026c;d;e), Gemini 3.1 Pro (Google DeepMind, 2026), Qwen3.6-Plus (Qwen Team, 2026a), Seed2.0 Pro (ByteDance Seed Team, 2026a), GLM-5 (GLM-5 Team et al., 2026), and MiniMax M2.7 (MiniMax, 2026) as reference generators. As discussed in the main text, however, model-generated pages can themselves converge on familiar design patterns. For the 100 atypical pages, we therefore pair a controlled design sampler with Claude Opus 4.8 (Anthropic, 2026d) to introduce less conventional visual combinations. The sampler combines 16 predefined design families (e.g., Swiss typography [7], brutalism, and terminal interfaces) with varied palettes and family-specific font stacks, using flat fills, square corners, and optional hard shadows. The model then refines the copy and page structure, while automated checks enforce the palette, font, and styling constraints. These models participate in constructing the reference pool, a role distinct from that of the nine systems subsequently evaluated on the benchmark.

From webpages to task evidence. The stored webpages are construction assets, but the references shown to agents consist only of images. To produce these visual inputs, Playwright renders each page at a 1920×1080 viewport and captures scripted scrolling and interaction states, yielding 5,522 frames across the pool. For conventionally styled synthetic pages, we retain their original page-specific demonstration scripts. For template-derived and atypical pages, we generate scripts from the measured page height and available hover targets (identified through known selectors or DOM probing), interleaving stepwise scrolling, hovers, and short pauses. We retain the capture-action script and a time-stamped frame manifest for each page so that the resulting images remain associated with the actions and source that produced them. When references are assembled into a task, frame order and source-page boundaries are preserved. The agent can thus distinguish views of the same page from views of different references without receiving their implementations. Before a request can be written, we also need to establish which reference properties these images actually expose. Gemini 3.1 Pro (Google DeepMind, 2026) profiles the frames along the six visual dimensions in § 3.2, providing a description of the visible evidence and a `rich`, `weak`, or `none` label for each dimension. These profiles connect reference collection to query synthesis: they restrict the assignment of reference-dependent requirements to properties supported by the supplied images, rather than by information available only in the underlying webpage.

B.2 Controlled Task Construction

We construct each user request from a structured recipe that combines the reference-use specification in § 3.3 with choices about the target webpage and how the user expresses their needs. The target domain is drawn from a taxonomy of 8 broad categories, 40 subcategories, and 206 site types, ranging from productivity tools such as kanban boards to salon booking pages and API documentation. The recipe also selects one of 14 user profiles, each defining an identity, priorities, and speaking style. For example, a shop owner may emphasize contact details in everyday language, whereas a front-end engineer focuses on semantic structure and responsive behavior.

The recipe also controls the intended page's functional complexity, content density, and explicit constraints. Functional complexity ranges from static pages to stateful applications and data visual-

[7] https://en.wikipedia.org/wiki/Swiss_Style_(design)

ization. Up to three tags specify requirements concerning audience, presentation, content volume, delivery, tone, or functionality. These can call for mobile-first display, long-form text, image-led content, offline operation, or bilingual switching. Concrete visual requirements, such as body text of at least 18px, a specified brand color, or square-cornered buttons, are recorded independently of the references.

Separately, we sample the language, form, and level of detail of the request. Requests are written in Chinese or English and take the form of flowing prose, casual messages, a flat list of requirements, or a structured product requirements document (PRD) with section headings and acceptance criteria. The level of detail is sampled independently of the page's functional complexity. It ranges from a single overall need, through a brief description of three to five sections, to detailed content, data, and interactions for each section. Target length depends on both form and level of detail and is measured in Chinese characters or English words. Two exemplars matching the request's language and form guide organization and tone without supplying topics or wording to reuse.

Gemini 3.1 Pro then converts the recipe and ordered screenshot groups into a natural-language request using the prompt in Fig. 10. The prompt asks the model to identify each source by its appearance or position and explain how its selected properties should inform the design. Each reference is used only for the specified aspects of the page, and its content is not reused. Explicit values, such as the 18px minimum for body text, must be preserved so that constraints remain checkable after rewording. Visual descriptions must be grounded in the screenshots; motion that is not visible in them is phrased as a new user requirement. We retain the resulting request with its recipe for keypoint construction. Fig. 8 shows three examples with different forms and user profiles.

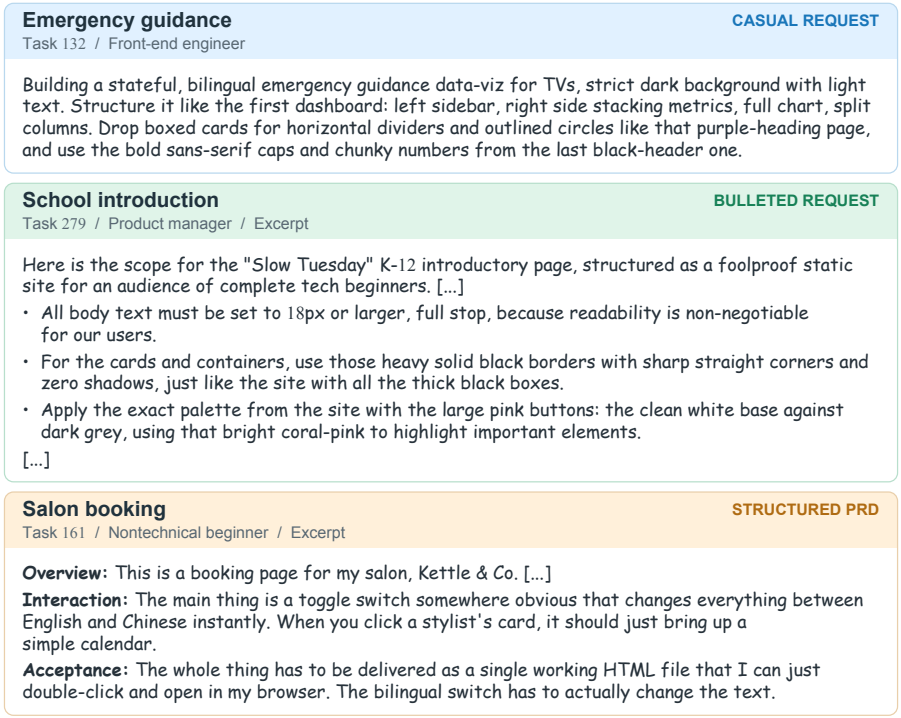


Figure 8: Generated user query examples. English requests from the final task set illustrate casual, bulleted, and structured PRD forms. Task 132 is shown in full; Tasks 279 and 161 are excerpted, with `[...]` marking omissions. Original wording is retained.

B.3 Keypoint Construction and Quality Assurance

Keypoint construction must preserve the reference relationships established during query synthesis while also covering the content and functionality expressed in the completed request. We handle these two parts separately. Reference assignments and explicit textual constraints are converted programmatically into fixed items from the structured specification, retaining their intended sources and requirement types. Claude Opus 5 (Anthropic, 2026f) then reads the user request and adds

keypoints for content, structure, navigation, and functionality without rewriting the fixed items. This division keeps the specified reference relationships intact while allowing the keypoints to cover requirements that emerge in the natural-language request.

To make the added keypoints assessable, the construction prompt (Fig. 12) links each criterion to both the user request and the evidence needed to evaluate it. Each item must first cite the supporting request text and clarify what it checks: the existence of an element, the correctness of its implementation, or the precision of its details. Distinguishing these levels prevents the mere presence of an element from being treated as sufficient evidence that it has been implemented as instructed. The resulting criterion must then be answerable from the evidence available during evaluation, ruling out questions requiring source-code inspection or unavailable timing measurements. For interaction-dependent requirements, this also means identifying both the control and the action needed to reach the relevant state, so that the subsequent browser-capture process can expose the behavior being assessed.

The generated keypoints are then reviewed by GPT-5.6 Sol (OpenAI, 2026c) with high reasoning effort and the prompt shown in Fig. 13. The review separates completeness from item validity: a global pass first proposes missing requirements, after which a local pass examines individual items and returns `keep`, `modify`, or `remove`, together with a reason. Both passes use the reference-use specification to check whether the criteria preserve the intended task. In particular, an ignored dimension must not introduce a requirement, an adaptation must not become an exact-match demand, and an inversion must not reward copying. These checks address errors that could otherwise make a correct implementation appear unsuccessful. Generation and model review use a configured maximum output of 128,000 tokens. Finally, the expert review described in § 3 revises the keypoints before they are used for evaluation. Fig. 11 presents the instruction template for expert keypoint review.

Final task set. The final benchmark contains 300 tasks, each pairing a user request and ordered reference screenshots with a verified set of keypoints. Together, these tasks use 534 distinct webpages in 972 reference assignments and expose 3,491 reference frames. Stable task identifiers and input hashes preserve the correspondence between each request, its references, and its evaluation specification when these components are passed to the execution and judging stages.

B.4 Standardized Agentic Execution

With the task components fixed, execution exposes only the information intended for the evaluated agent: the user request, ordered reference images, and an environment brief. The verified keypoints remain hidden and are used later by the evaluator. All released runs use OpenCode 1.18.25 with the same 100-turn cap, implementing the shared coding-agent setting described in § 4.1.

The environment brief is matched to the language of the request and explains both the required deliverable and the available development workflow. It asks the agent to "write the finished page to `index.html`," requires a single HTML file, and states that "grading renders the page offline." It also describes the inspection loop explicitly as "draft → screenshot → compare → fix → screenshot again." To make this loop executable, the workspace provides shell and file-editing tools together with a Playwright-based `shot.py` helper. The helper renders the current HTML at widths of 1920, 768, and 390 pixels and returns image paths along with page height, console errors, and attempted external requests. These outputs allow the agent to inspect its rendered implementation and identify problems before revising the page.

Network access remains available during development, but it does not relax the offline submission requirement. Dependencies must be inlined in the final page so that its evaluation does not rely on external resources. Likewise, the availability of development tools does not grant access to benchmark data, hidden criteria, or original reference implementations. Such access is denied and audited. At completion, we retain one official artifact per model–task pair, together with a run record containing the enforced limits, stop reason, transport attempts, token usage, and artifact hashes. This record links the webpage evaluated downstream to the trajectory and resource use reported for that run.

B.5 Evidence Collection and Dual-Track Judging

The agent's inspection tools support development, but the evidence used for scoring is collected separately from the submitted `index.html`. This distinction ensures that the evaluator examines the final webpage through its own capture and measurement procedure rather than relying on the views the agent happened to inspect. The hidden keypoints guide this procedure by identifying the states and properties that need to be exposed. The resulting evidence is then passed to the programmatic and VLM judging tracks described in § 4.3.

Capture and measurement settings. Playwright/Chromium first renders the artifact at 1920×1080, with a target of five to ten distinct primary frames. Since requirements can concern states that are not visible on initial load, a deterministic action planner uses the keypoints to plan at most eight states to expose. For each item, it tries up to four candidate controls and allows 900 ms for a state transition. These settings govern how the browser attempts to obtain the interaction evidence specified during keypoint construction.

The evaluator also measures the page at widths of 1280, 768, and 390 pixels to assess behavior beyond the primary desktop view. Additional stress fixtures test robustness when content differs from the initial presentation: eligible collections are replaced with zero or 300 rows, an 80-character unbroken string is injected, and mixed CJK/emoji content is supplied. The resulting measurements complement the ordinary rendered views by exposing behavior under narrow layouts and abnormal content. If a measurement is unavailable or inapplicable, that status is recorded explicitly. The absence of a usable measurement is not treated as evidence that the corresponding test succeeded.

From evidence to judgments. For program-verifiable keypoints, the collected browser measurements determine pass/fail verdicts. Requirements that need semantic or perceptual interpretation are instead assigned to Gemini 3.1 pro, configured with high thinking level and a maximum output of 65,535 tokens. This judge receives visual and measured evidence rather than the submitted HTML, reference source code, or solver identity. Its task is to assess what the webpage demonstrates against the stated requirement, rather than infer success from implementation claims.

The perceptual judge processes batches of 5 keypoints, with a separate assessment for each item. Each batch contains the user request, the supporting basis for each item, the reference assignments, and normalized browser facts. To keep these facts and images interpretable, the images follow a fixed order: reference frames, up to ten main artifact views, up to two views per narrow viewport, and up to two stress-fixture views. An accompanying manifest identifies the groups, making clear which images describe the reference and which show the submitted page under each evaluation condition.

Before assigning a rating, the judge must record the `expected` requirement, the `observed` evidence, and the `deviation` between them. This sequence asks it to establish a concrete comparison before choosing a score. Fig. 14 presents these audit instructions and the rating rubric. The full template also specifies evidence handling and exceptions for unavailable external inputs. The resulting item scores are combined with core-function evidence using the aggregation rule in § 4.3. We retain render manifests, measurements, action outcomes, and item verdicts separately, allowing the recorded judgments to be inspected and the scores to be re-aggregated without another model call.

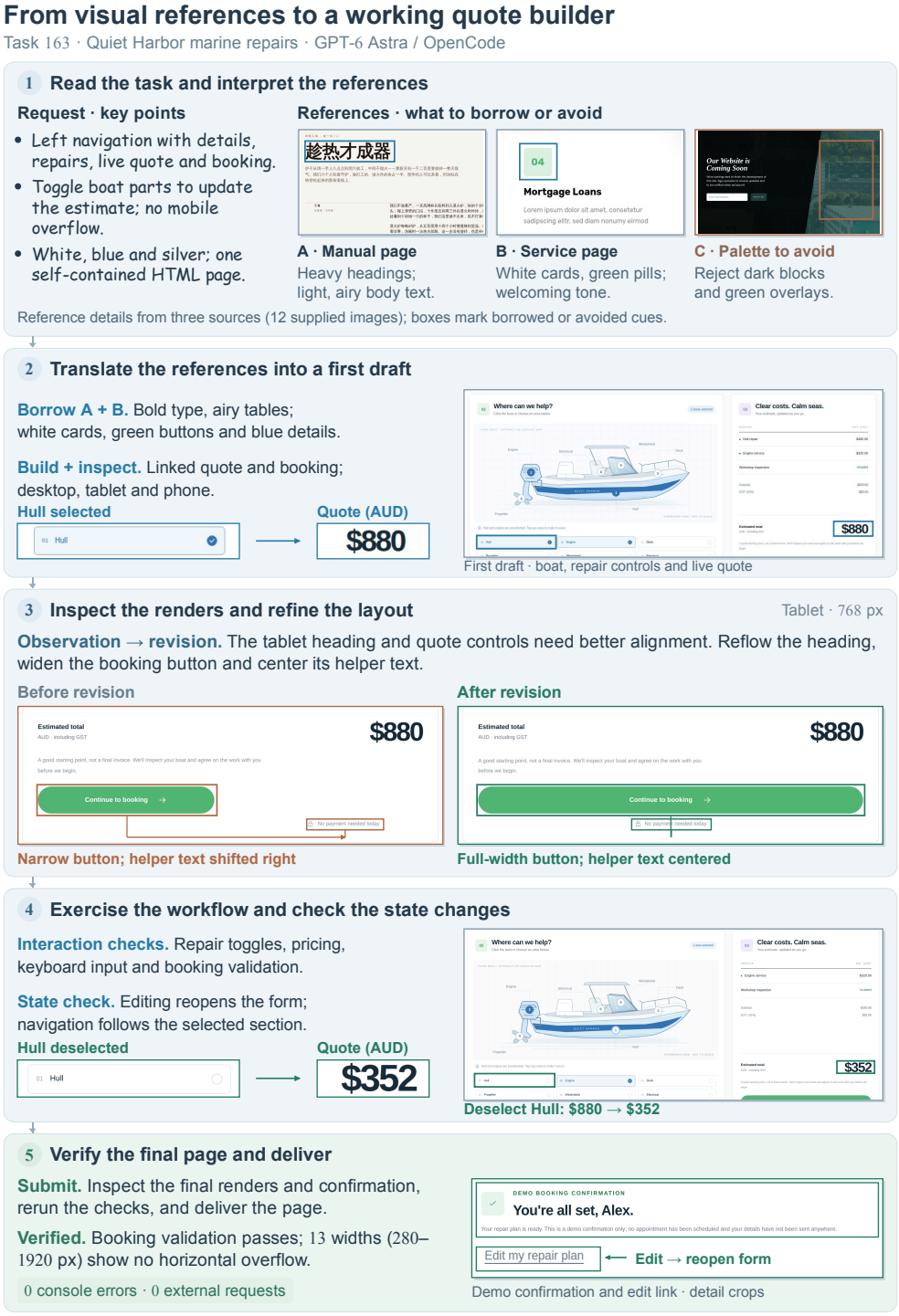


Figure 9: An end-to-end development trajectory on Task 163. Boxes and enlarged screenshot crops highlight reference cues, responsive-layout revisions, and changes in the quote and booking state.

Task-writing prompt: multi-reference requests

You are building test prompts for a "Visual Reference Webdev" benchmark.
[. . .]
You are given a RECIPE. Follow every line of it, but the finished text must read like something a person actually typed — never like a filled-in form.
[. . .]
The attached screenshots are **{n_groups} different websites**, in this order: {group_intro}
[. . .]
{mix_block}
[. . .]

1. **Every single thing you say about these pages must be pinned to the page it applies to** — what you want taken from one, and equally what you saw on one and do not want. A reader holding the screenshots has to be able to tell, for each of your asks, which page you meant.

[. . .]

2. **Never ask for a page to be followed as a whole.** No "basically make it look like this one", no "follow the first one overall", no "same vibe as that page overall". You are cherry-picking, and a blanket instruction would silently override your own picks. If you catch yourself writing a sentence that gives one page general authority over the design, cut it.

[. . .]
Only claim what you can actually point at in the screenshots. This is the single most damaging mistake you can make: a detail you invent becomes a test question nobody can answer, and a model that gets it right gets marked wrong.

- Before writing any visual detail — corner radius, shadow, border, weight, contrast, density — find it in the right page's screenshots first. If you cannot point at it, do not write it.
- **Attributing a real detail to the wrong page is just as bad as inventing one.** Read the notes above and keep them straight: the colours you liked are on one page, the cards on another.

[. . .]

2. **The specific value has to survive.** Reword the sentence however suits you, but the actual thing — that typeface, that colour, that size, that many columns, those corners — must still be there and still be checkable by someone who never sees the screenshots.

[. . .]
Write in **{lang_name}**. Form: **{form_name}**. {form_spec} Target length: about {length_lo}-{length_hi} {length_unit}.
[. . .]

6. Output ONLY the JSON object below — no markdown fences, no explanation.

{{"user_prompt": "<the instruction, in {lang_name}, in the required form>"}}

Figure 10: Prompt for user-query generation. Across all prompt figures, [. . .] denotes omitted passages. Formatting and line breaks are adjusted for presentation.

Expert keypoint review

Your role. Review and revise the evaluation checklist so that it faithfully represents the user's request and can be applied to different valid implementations. You are reviewing the evaluation criteria, not scoring a generated webpage. Do not use model outputs or scores to decide which requirements belong in the checklist.

Materials. Read the user instruction, ordered reference screenshots, reference-use specification, and draft checklist, including each item's identifier, supporting basis, and annotations. Review both seeded and model-generated items.

1. **Check coverage against the task.** Identify the requested content, structure, navigation, visual properties, and essential behavior, then check whether the checklist covers them. Add missing requirements with a supporting basis. If a function is implicit but necessary to fulfill the stated task, explain that necessity explicitly. Do not add personal design preferences or optional conventions.
2. **Ground each item in the correct evidence.** Quote the relevant request text and, for a reference-dependent property, identify the source page and frame supporting it. Confirm that the cited feature is visible and belongs to that reference. Remove invented details and unsupported requirements. If the request, screenshots, and specification conflict, flag the inconsistency rather than silently changing the task.
3. **Preserve the intended reference relationship.** Check each constrained dimension against its assigned requirement type:
   - `copy`: assess the requested property against the assigned reference.
   - `adapt`: assess the selected characteristics and requested transformation, without adding exact-match demands.
   - `invert`: assess the requested departure from the reference, not similarity to it.
   - `specify`: follow the explicit textual constraint rather than importing a reference property.
   - `ignore`: do not introduce a requirement for the unconstrained dimension.
4. **Make the criteria specific and verifiable.** Each item should assess one requirement without duplicating another. Distinguish existence, correctness, and precision, and do not invent numerical tolerances. For interactions, name the control, action, and observable outcome. Check that the criterion can be judged from rendered views or available browser measurements, not source-code claims or unseen behavior. Use programmatic checks where reliable measurement is available and perceptual judgment otherwise. Flag incorrect source, dimension, depth, importance, or judging-track annotations.
5. **Return an auditable revision.** Mark each existing item as `keep`, `modify`, or `remove`, and label new items `add`. Preserve existing identifiers. For every change or addition, provide the revised wording where applicable, supporting evidence, corrected annotations, and a brief reason. Return the revised checklist with a separate list of unresolved ambiguities or missing evidence. Do not approve unresolved items by assumption or change the scoring rubric and aggregation rule.

Figure 11: Instruction template for expert keypoint review. Reviewers check coverage, evidence grounding, reference-use consistency, and verifiability, then return revised keypoints with reasons for changes and unresolved issues.

Keypoint-construction prompt

Role
You are a benchmark checklist construction expert. Items covering **reference style / source attribution / negative-reference requirements / explicit hard constraints** have already been generated programmatically from the task request (listed verbatim below). **Your only job is to add the content / structure / navigation / functional requirements in the user instruction.**
[...]
Reference-following directive matrix (context only: items for these dimensions are already generated)
{{ref_directive_matrix}}
Programmatically generated reference items (read-only: do not duplicate)
{{seeded_items}}
[...]
Rule 1: Only generate content/structure/navigation/functional items (ref_aspect must always be "")

- Every explicit **content / structure / functional** requirement in the user instruction must have a corresponding item.

[...]

- **Do not** generate items for aspects not mentioned in the instruction, even if they are prominent in the screenshots.

[...]
Rule 3: Three levels of depth, not just existence
Cover at least three levels for each section/function:
Level 1 — Existence ($\leq$20% of items): Does the element exist?

- "Does the page include an appointment form?"

Level 2 — Correctness (~40% of items): Is the element implemented correctly as instructed?

- "Does the navigation bar contain these four links, in this order: Home, About, Services, Contact?"

Level 3 — Precision ($\geq$40% of items): Do the content details match the instruction exactly?

- "Does the appointment form contain these four fields: name, phone, date, and time slot?"

[...]
Rule 5: basis must quote the original instruction
Each `basis` must quote the corresponding requirement from the instruction, giving the judge clear evidence.

- Bad basis: "The instruction asks for a form."
- Good basis: "The instruction states: 'The appointment form must include name, phone, and appointment date.'"

[...]
The checklist auditor only has **rendered screenshots + programmatically measured numerical facts** (overflow pixels, node counts, dominant colors, font families, counts of DOM changes before and after clicks, anchor-landing counts, and counts of numerical changes during loading). **The auditor cannot see the HTML/CSS/JS source** (deliberately: source code could claim "the reference color scheme has been fully followed").
[...]

- △ **An item about a particular state must name, in its question, how to reach that state**: screenshots show only the initial page state by default. The program clicks and captures the resulting frame only when the action is specified.
  - × "After expansion, are both the abstract text and citation entries shown?" (What expands, and how? This frame cannot be captured.)
  - ✓ "After clicking the 'Details' button on the right of a paper card, does the expanded area show both the abstract text and a BibTeX citation block?"

Figure 12: Keypoint-construction prompt excerpts.

Two-stage keypoint-review prompts

You are a quality auditor for evaluation checklists. Review a checklist generated by another model and check whether any important items are missing.

[...]

1. **First check coverage against the matrix**: the matrix is authoritative on whether these items should exist.
   - Dimensions marked `copy`: Are there enough **specific** items (stating what the screenshots actually show)? Add any that are missing.
   - Dimensions marked `adapt`: Are there **direction/tone** items? Add any that are missing, but do not add exact-color or exact-spacing requirements.
   - Dimensions marked `invert`: Are there items checking **differences in the required direction** (copying the screenshot should score poorly)? Add any that are missing.

[...]

- Dimensions marked `ignore`: **There should be no items at all, and you must not add any.**

2. **Then check coverage against the instruction**:
   - Every content/structure/functional requirement explicitly mentioned in the user instruction.
   - Implicit but necessary functions (e.g., an "e-commerce website" implies product displays, prices, etc.).

[...]

5. **Do not invent additions**: only add items genuinely required by the user instruction or screenshots and allowed by the matrix. Do not introduce dimensions the user did not address.

You are a quality auditor for evaluation checklists. Review the checklist item by item and determine whether each item is reasonable.

[...]

1. **Consistent with the matrix? (Most important, taking priority over all other checks)** An item's `ref_aspect` identifies the screenshot dimension it draws on. Check the matrix:
   - Dimension marked `ignore` → the item **should not exist**. Mark remove, with reason "violates the ignore directive".
   - Dimension marked `invert` → the item must check **explicit departure from the screenshot**. If it asks "Does it match the screenshot / reuse the screenshot's XX?", **the direction is reversed** (rewarding copying and penalizing correct behavior) → mark modify, provide a correctly directed version in `modified_question`, and state "what the screenshot shows + the direction requested by the user" in `modified_basis`.
   - Dimension marked `adapt` → only check direction/tone. If the item demands **exact** colors, pixel spacing, or font sizes, → mark modify and relax it to a directional question.
   - Dimension marked `copy` → the item should be specific (state what the screenshots actually show). Handle overly generic items under check 2.
   - Empty `ref_aspect` but the item clearly tests screenshot style → mark modify and supply the correct dimension name in `modified_ref_aspect`.
2. **Grounded in evidence?**
   - Does the item's basis have clear support in the original user instruction?
   - If the basis cites a screenshot (e.g., "the screenshot contains XX"), is that feature actually present?
   - If it is in neither the instruction nor the screenshots → **hallucination**. Mark remove.

[...]

Output format

Return a JSON array (no Markdown code block). **Provide one judgment for every input id**, without omissions.

[...]

- Do not add new items. Only make keep/remove/modify judgments.

Figure 13: Keypoint-review prompt excerpts. The global pass checks coverage, and the local pass reviews individual items against the reference-use specification and supporting evidence.

Perceptual-judging prompt

[...]

For each item, determine the following three things in order and include them in the output:

1. **expected (required value)**: Extract a **specific, verifiable requirement** from `basis`. Write a value, not a paraphrase.
   - ✓ "5 cards" / "dominant color #1a2340 (from the reference screenshot header)" / "exact text: 'Live leaderboard'"
   - × "Colors must match the reference" (this restates the requirement rather than giving a value).
2. **observed (measured value)**: The **actually measured value of the same dimension** in the **fact block**/rendered frames. This must also be a value.
   - ✓ "3 cards (see frame 04)" / "#4a90d9 (first entry in fact block colors.bg_by_area)" / "text reads 'Live rankings'"
   - × "Implemented correctly" / "Meets the requirement" (these are conclusions, not measurements).
   - If the value cannot be measured from any supplied material → write "cannot confirm" in observed.
3. **deviation (difference)**: The difference between expected and observed. **This field must not be omitted.**
   - Write "none" only when there is no difference at all. Writing "none" commits you to proving character-by-character/value-by-value equality.
   - Otherwise, specify the difference: "count differs by 2 (3/5)" / "similar hue but brightness differs by about 40%" / "text has been reworded".

[...]

Derive the score from deviation (do not form an impression and then justify it)

Complete expected / observed / deviation **first**, then use deviation to select a score from the table below. Do not reverse this order.

Use a **four-level 0-1-2-3 scale** (poor / fair / good / excellent). A score of 3 is extremely difficult to earn: it requires perfect, indisputable evidence.

Level / Score	Meaning and corresponding deviation
Poor **0**	Entirely missing, entirely inconsistent, clearly deviating (wrong count, reworded/summarized text, wrong position/hierarchy, clearly mismatched attributes) *Corresponding deviation:* One or more clear deviations, or complete absence
Fair **1**	Correct direction with acceptable deviations (roughly correct but imprecise details, right color family but different values, slightly too few items, minor layout differences) *Corresponding deviation:* Quantifiable deviations, but recognizably the same overall design
Good **2**	Essentially correct, with only tiny imperfections (color difference <5%, spacing difference 1-2 px, almost indistinguishable) *Corresponding deviation:* Tiny or no deviation, but not verifiable value-by-value perfection
Excellent **3**	**Perfect match**: expected and observed are equal character by character/value by value, with zero deviation and sufficient evidence *Corresponding deviation:* deviation is genuinely "none", and both values can be checked character by character

[...]

Figure 14: Perceptual-judging prompt excerpts. The judge must record expected and observed values and their deviation before applying the four-level rubric.