APPWRIGHT: BENCHMARKING END-TO-END APPLICATION CONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents are increasingly deployed as products that build *complete applications* from natural-language requirements, yet existing benchmarks evaluate function-level correctness, repository-level issue fixing, or single-page web generation judged by human preference. None measures the end-to-end ability to deliver a full, runnable, and *well-designed* application across the forms industrial users request. We present APPWRIGHT, a benchmark abstracted, under a desensitization and deduplication pipeline, from the request traffic of a production no-code application-generation service, covering five categories: web apps, tools, games, mini programs, and mobile apps. APPWRIGHT provides (i) 184 tasks from anonymized real user requirements, each shipping the original query and a structured Product Requirements Document (PRD) with an acceptance rubric and a difficulty label (D1–D4); (ii) a reproducible construction-and-execution harness reimplementing the Code Arena specification; and (iii) a reward agent scoring each delivered application on two orthogonal axes—*usability*, verified by actively operating the deployed app, and *aesthetics*, a versioned nine-item rubric distilled from production design reviews. To our knowledge, APPWRIGHT is the first benchmark to jointly evaluate usability and aesthetics under structured, reproducible criteria. Across nine frontier models the two axes disagree: the top model differs by metric, difficulty rather than application category decides the ranking, and aesthetics discriminates only as a gate—so a single aggregate score would hide the differences that separate systems. We release the task suite, harness, and reward agent.[1]

1 INTRODUCTION

Coding agents are moving from completing code to delivering products. Where early systems were judged on autocomplete and single-function synthesis, current agents are deployed as application builders: a user states a requirement in natural language and the agent is expected to return a *running application*, not a diff or a snippet. Commercial no-code services have made this the primary consumption mode of code models (Google and others, 2025), and the delivery unit has shifted accordingly: an application must build, launch, respond to a user's actual workflow, and look like something a user would accept. Evaluating such a system by any isolated slice of that chain—the code it writes, the page it renders, or the impression a rater forms in a few seconds—answers a different question than whether the product was delivered.

Existing code-generation evaluations fall into three levels, none of which answers that question. *Function/snippet-level* benchmarks such as HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), LiveCodeBench (Jain et al., 2024), and BigCodeBench (Zhuo et al., 2024) measure algorithmic correctness of isolated code; *repository-level* benchmarks such as SWE-bench and its variants (Jimenez et al., 2024; Yang et al., 2024) start from a mature codebase and evaluate issue localization and repair, not from-scratch construction; *arena-level* evaluations such as Code Arena (Arena, 2024) and WebDev Arena (LMArena, 2025) rely on pairwise human preference over mostly single-page web apps, without structured acceptance criteria and without coverage of mini programs or mobile apps. Recent application-generation benchmarks close part of the gap but each covers a slice—

[1]Code: `https://github.com/Azure99/app_wright`. Data, trajectories, and application information: `https://huggingface.co/datasets/dongdaxiang/app_wright`.

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.

Vision2Web (Vision2Web Authors, 2026) and WebCoderBench (Liu et al., 2026b) stay within web deliverables, MiniAppBench (MiniAppBench Authors, 2026) within assistant-returned interactive HTML, and Cookie-Bench (Cookie-Bench Authors, 2026) and WorkBuddy (Tencent WorkBuddy Team, 2026) add verification stages without spanning the full set of application forms industrial users request (§2).

A delivered application is accepted on two conditions, applied separately: it must *work*—the stated task can be completed through normal interaction—and it must *look right*—the result is presentable enough to show to someone else. Existing evaluations capture at most one, and usually indirectly. Functional benchmarks report pass rates over checkable behavior and say nothing about presentation; design-quality instruments such as VisAWI (Moshagen & Thielsch, 2010) and learned scorers like UIClip (Wu et al., 2024) measure visual quality but are administered to artefacts rather than wired into a construction pipeline; and arena-style leaderboards (Arena, 2024; LMArena, 2025; Design Arena, 2025) compress presentation, functionality, and content into one human preference label. To our knowledge, APPWRIGHT is the first coding-agent benchmark to treat usability and aesthetics as two *independently judged, rubric-based* axes over the same artifact, evaluated jointly but never fused into a single aggregate.

We introduce APPWRIGHT, abstracted from the request traffic of a production no-code application-generation service, which grounds it in a realistic requirement distribution, production-validated acceptance criteria, and a real build pipeline abstracted into a reproducible harness rather than a toy environment. Each of the 184 tasks preserves both the original user query and its structured PRD, carries a difficulty level (D1–D4) and an implementation form, and spans five application categories; applications are scored by a dual-axis reward agent along independent usability and aesthetics axes (§5).

Our contributions are: **(1)** a validated dual-axis reward agent that operates the deployed application and scores usability and aesthetics as *independent* policies, checked against human judgment at agreement above 80% and precision above 90% (§5); **(2)** a cross-model evaluation with that agent—nine frontier reasoning/coding models ranked on both axes, showing the axes are not redundant and that open-weight models now exceed some closed ones (§6); **(3)** a five-category task suite of 184 production-derived tasks spanning web app, tool, game, mini program, and mobile app, stratified by category, difficulty, and implementation form (§3); and **(4)** a reproducible application-generation harness reimplementing the Code Arena specification, released with the suite and agent so every score is attributable to the model rather than the scaffold (§4). The protocol is not devised for the benchmark: it reproduces the product practice of a deployed no-code service, where automated judging is paired with a sampled human calibration pass (§7). Evaluating nine models under one fixed scaffold yields three findings, detailed in §6: no single metric identifies the best system—the excellent, usable, and aesthetic pass rates crown three different leaders; difficulty, not application category, sets the ceiling; and aesthetics discriminates only as a binary gate, not as a mean. The two axes disagree precisely where a single aggregate would hide the difference.

What sets APPWRIGHT apart is a conjunction of constituents: production provenance, a per-task structured PRD whose feature points are individually checkable, acceptance criteria the deploying service applies to its own output, and two human-validated rates for usability and aesthetics. Each has a near neighbor in prior work; no existing application-generation benchmark provides all four at once (§3.2).

2 RELATED WORK

Code generation, repository tasks, and application construction. Early benchmarks evaluate the correctness of isolated code—HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), LiveCodeBench (Jain et al., 2024), BigCodeBench (Zhuo et al., 2024)—and saturate as models improve, while SWE-bench (Jimenez et al., 2024) and its multimodal extension (Yang et al., 2024), along with terminal-oriented suites (Terminal-Bench Team, 2025), evaluate issue resolution over existing repositories. Both families measure correctness against a supplied specification—a function signature, or an existing codebase—rather than from a blank product requirement, and neither asks whether the artifact is presentable. A growing body of web/app generation work moves closer on three axes. *Design-to-code* benchmarks (Design2Code (Si et al., 2025), DesignBench (DesignBench Authors, 2025)) score similarity to a target screenshot, which is reference following rather

than open-ended construction. *Open-ended* benchmarks differ in coverage and verification: Vision2Web (Vision2Web Authors, 2026) and WebCoderBench (Liu et al., 2026b) remain within web deliverables, MiniAppBench (MiniAppBench Authors, 2026) covers assistant-returned interactive HTML, FullStack-Bench (FullStack-Bench Authors, 2026) and PlayCoder (PlayCoder Authors, 2026) probe full-stack and playable-code construction, and a further group adds a verification mode or visual dimension over a recognized slice of this space (VISTA Authors, 2026; Lu et al., 2025; Wang et al., 2026; ByteDance Web-Bench Team, 2025; FrontendBench Authors, 2025; Liu et al., 2026a; InteractWeb-Bench Authors, 2026; Zhang et al., 2025; WebCompass Authors, 2026; Cookie-Bench Authors, 2026; Tencent WorkBuddy Team, 2026). *Arena-style* platforms (Arena, 2024; LMArena, 2025; Design Arena, 2025; Jung et al., 2025; AfterQuery, 2025; Google and others, 2025) collect holistic human preference but lack diagnostic rubrics and compress functional, content, and visual quality into one label. Across all three axes, visual quality is absent or entangled with functionality, and no prior benchmark reports it as a separately judged rubric-based rate over the same delivered application (Table 9, Appendix I).

Judging, reward modeling, and design-quality instruments. LLM-as-judge reliability (Zheng et al., 2023) and its validation for web development (Li et al., 2026) motivate careful judge design; our reward agent is an interactive, dual-axis judge that operates the deployed application rather than scoring static screenshots or code, is validated against production human standards, and is released as a component reusable as a training reward signal. Psychometrically validated scales—VisAWI (Moshagen & Thielsch, 2010; 2013)—and usability/accessibility instruments—UEQ (Laugwitz et al., 2008), SUS (Brooke, 1996), WCAG (W3C, 2023)—establish the design-quality and usability constructs we adopt, but are administered to human raters rather than wired into a generation pipeline; learned models such as UIClip (Wu et al., 2024), designer-rated datasets such as UI Quality Bench (Revelo, 2026), and rubric-based visual scoring in OpenDesign (Xiao et al., 2026) inform our rubric. APPWRIGHT is, to our knowledge, the first coding-agent benchmark to make a versioned aesthetics rubric a first-class axis alongside usability.

3 THE APPWRIGHT TASK SUITE

3.1 DESIGN PRINCIPLES, FORMAT, AND DIFFICULTY

APPWRIGHT follows four principles. **(1) Authenticity:** tasks are abstracted from real production traffic through anonymization and stratified sampling, preserving genuine requirement diversity and the ambiguity of real user requests. **(2) Dual specification:** every task ships both the original user query q, which retains user intent and its level of detail, and a structured PRD p, which reorganizes that intent into an implementation-oriented specification (overview, target users, page hierarchy and features, business rules, exception handling, acceptance criteria, out-of-scope features). The PRD is what makes acceptance checkable: a bare query names a goal without pinning its features, so a judge scoring only against the query has no fixed list to verify against and must reward surface plausibility, inflating scores for applications that do not actually work. The PRD enumerates the feature points individually, letting the reward agent hold a delivered artifact to the requirement rather than to an impression of it; retaining both q and p additionally enables analysis of the gap between raw intent and engineered requirements. **(3) Difficulty stratification:** each task is assigned a level D1–D4 by a fixed-rubric LLM annotator, supporting fine-grained capability profiling. **(4) Operational acceptance:** each task ships a structured rubric whose feature points are individually checkable and consumed directly by the reward agent (§5).

Formally, a task is a tuple $t = (q, p, R_u, R_a, \ell)$: the natural-language query q, the structured PRD p, a usability rubric R_u (a checkable feature list derived from the PRD), an aesthetics rubric R_a (§5), and a difficulty level $\ell \in \{\text{D1}, \ldots, \text{D4}\}$. Some tasks additionally carry multimodal materials (e.g., reference images) or a "with-backend" flag requiring a working database. Difficulty is assigned by a fixed-rubric LLM annotator that scores two complementary axes: *structural scale* (S), the number and hierarchy of pages/modules and whether multiple independent role-facing ends exist, and *logic depth* (L), acceptance conditions, state transitions, cross-module constraints, algorithmic requirements, and real-time synchronization. A rule then maps (S, L) to D1 (compact structure, shallow logic), D2 (standard-scale application), D3 (high demands on logical rigor or code organization), or D4 (both large structure and deep business/system logic).

Table 1: Comparison with recent application-generation benchmarks. "Forms" counts distinct deliverable categories; "Difficulty" records whether tasks carry an explicit difficulty label. All listed benchmarks build and serve the artifact.

Benchmark	Tasks	Form	Difficulty	Evaluation
Vision2Web 2026 (ICML'26)	193	web	3 tiers	GUI agent verifier + VLM judge
WebCoderBench 2026b (ACL'26)	1,572	web	none	24 rule/LLM metrics
FullStack-Agent 2026 (ICML'26)	–	full-stack	none	front/back/db test cases
MiniAppBench 2026 (ICML'26)	500	HTML	3 tiers	agentic browser test
PlayCoder 2026 (FSE'26)	43	GUI / games	none	LLM playthrough agent
Cookie-Bench 2026	1,000	web	3 tiers	3-stage agent
WorkBuddy 2026	260	4 domains	none	tests / rubric / rules
APPWRIGHT (ours)	184	**5 forms**	D1–D4	dual-axis reward agent

3.2 COMPARISON WITH RECENT APPLICATION-GENERATION BENCHMARKS

Application-generation benchmarks have begun to appear; Table 1 states what distinguishes APPWRIGHT from them, and the contrast is structural. On coverage, each recent benchmark commits to one deliverable form—web pages (Vision2Web, WebCoderBench), full-stack sites (FullStack-Agent), assistant-returned interactive HTML (MiniAppBench), or desktop GUI applications (PlayCoder)—whereas APPWRIGHT spans five forms under one scaffold, so cross-form comparison happens within a single run. The distinction in scoring is not the machinery—we too drive a browser and read screenshots, as PlayCoder's playthrough agent and Vision2Web's verifier do—nor the number of axes, since FullStack-Agent, PlayCoder, and Vision2Web all report several. It is the per-task grounding: what a task ships and where its criteria come from. Most benchmarks ship a bare query and score against a reference or a test suite not derived from the artifact's users—Vision2Web and PlayCoder against crawled references, FullStack-Agent and WorkBuddy against supplied tests, WebCoderBench, MiniAppBench, and Cookie-Bench against author-defined metrics. Every APPWRIGHT task instead ships a structured PRD alongside the query, so acceptance is a fixed list of checkable feature points rather than an impression and a plausible-looking but non-working application cannot score high (Principle 2); and where the judged dimensions elsewhere are defined by the authors, APPWRIGHT's requirements and rubrics are abstracted from production request traffic and are what the deploying service applies to its own output. Finally, usability and aesthetics—the two constructs an application is actually accepted on—are reported as two independent human-validated rates, neither a fidelity score against a supplied reference (Vision2Web's two scores match a prototype, FullStack-Agent's three run supplied tests). Sample scale runs the other way—WebCoderBench draws 1,572 requirements, an order of magnitude above our 184—trading breadth for the D1–D4 stratification and per-category balance a smaller suite can support.

3.3 FIVE APPLICATION CATEGORIES

APPWRIGHT covers five categories, partitioned into three groups: browser-delivered software, mobile-side deployment targets, and interaction-intensive games. **Web** apps run in the browser (corporate/product sites, content platforms, dashboards, multi-page admin systems). **Tool** applications center on a specific function (transform/compute/analyze, content generation, interactive demos), and also absorb lightweight mobile-browser or embedded-WebView (H5) flows built around a single shareable business task. **Mini Program** and **Mobile App** make up the mobile side: the former targets the mini-program runtime (primarily WeChat) with its interaction, navigation, and deployment constraints, the latter smartphones with touch-appropriate interaction and optional device capabilities (camera, location, local services). **Game** applications are built around explicit mechanics (scoring, timing, progression, levels, win/lose conditions); because they are interaction-intensive (real-time input, animation, state-heavy loops), we separate them from the other four as their own category, so that performance on this demanding form can be drilled into rather than averaged away.

This taxonomy lets us decompose application-generation ability both by deployment target and by interaction demand, rather than relying on a single aggregate score.

The joint app-type $\times$ difficulty distribution is given in Appendix Table 10. The median query and PRD lengths are 655 and $2{,}484$ characters (405 and $1{,}587$ tokens under the GLM-5.2 tokenizer), a median per-task PRD/query ratio of 4.98 by characters and 4.86 by tokens: the PRD does not merely restate the query but expands intent into page structure, feature logic, exception handling, and acceptance conditions.

3.4 Implementation Form

Orthogonal to application type and difficulty, each task is tagged with one of three *implementation forms* derived from the interaction and data-persistence requirements in its PRD, capturing infrastructure dependency rather than difficulty. **Native single-HTML** apps (48; 25.7%) implement their core functionality in a single HTML file with native HTML/CSS/JavaScript and no backend database. **Complex frontend** apps (69; 36.9%) require richer page organization, component interaction, or client-side state management but still no backend database. **Backend-persistence** apps (70; 37.4%) need server-side storage for shared data or cross-user flows; tasks relying on browser-side storage (LocalStorage, IndexedDB) are not counted here. Backend-persistence apps concentrate in Web (33) and Mini Program (21), complex-frontend apps in Game (23) and Tool (20).

3.5 Construction Pipeline and Quality Control

AppWright is built from a six-month window of user queries drawn from the request traffic of a production no-code application-generation service. Every query passes a desensitization review removing personal, organizational, and otherwise identifying content; the exposed text is the user's requirement statement only. Construction proceeds through five stages—semantic deduplication, query–PRD preparation, difficulty annotation, balanced downsampling, and compatibility filtering—under one principle: *preserve the user's intent while constraining the specification to the minimum functionality that fulfills the core goal*. The stage-by-stage procedure, the three generation controls (intent preservation, minimum scope, no clarification round-trip), the compatibility exclusions, and the contamination argument are in Appendix J.

This process yields 184 tasks (joint distribution in Appendix Table 10). The suite covers real-world requirements *within* the supported execution and evaluation scope; because of balanced downsampling and compatibility filtering, its proportions should not be read as the frequency of these types in production traffic.

4 Application Generation Harness

AppWright reimplements the Code Arena harness specification. The harness turns a task into a *sealed application artifact* through a strictly bounded, per-case rollout, and then serves or judges that artifact by restoring it from scratch. Two commitments define it: a minimal, framework-agnostic *model-visible interface* (the model sees only a template system prompt, the query and PRD, a whitelisted tool set, and redacted tool results), and a clean separation between rollout and evaluation, so that scoring never contaminates the recorded artifact. Both are what makes a score attributable to the model rather than to the scaffold or the judge.

Construction: a bounded, sealed rollout. Each case runs in a fresh isolated container from the template's starter workspace, with nothing carried over between cases. The model builds through the template's whitelisted tool set (13 tools for `react-vite-tailwind`), driven by a single OpenAI-compatible chat endpoint with `stream=true` and `parallel_tool_calls=false`; the first turn contains only the system prompt and the query–PRD user message, after which the conversation grows only by the model's own tool calls and the redacted results returned to it. There is no hidden planner, persistent session, or continuation, and no rubric, difficulty label, or reference solution is ever exposed to the model. The loop ends when a completion returns no pending tool call; a missing terminal reason, a `length`/`content_filter` finish, a build failure, or a failed readiness check all make the case unsuccessful. A returned "done" is therefore not success: what commits

a rollout is an atomically written `manifest.json` sealing the verified `source/runtime`. Because the image, starter workspace, template prompt, tool set, and budget are fixed, any rollout is reproducible from its record.

Evaluation: fresh restores, per-category runtimes. Serving and judging never reuse a live rollout process. Each restores the sealed artifact into a fresh runtime—the SQLite snapshot returned to a clean state for backend-persistence apps—and never writes back to the rollout, so evaluation is side-effect-free and repeatable. When a case needs both rewards, the outer flow runs them serially as *usability* $\rightarrow$ *teardown* $\rightarrow$ *aesthetic*, each from an independent restore sharing no app, browser, reasoning context, screenshots, or writable state; the aesthetic judge runs even when usability fails or is incomplete. This is why we report the two axes as separate rates rather than one aggregate (§5): the judges cannot see each other's evidence. Templates bind each category to a runtime and an acceptance procedure, and "does it run" is itself a judged hard gate: browser-runtime templates (web, tool, game) build and serve the `dist` bundle under a fixed Playwright/Chromium stack; mobile-app templates add smartphone-viewport emulation; mini-program templates route through a mini-program simulator. Like the Code Arena convention it follows, the harness ports models at low cost in both directions.

Tool schemas, path-isolation and redaction rules, template budgets, runtime versions, the mini-program simulator setup, and the released image tag are specified in Appendix C.

5 Dual-Axis Reward Agent

Unlike evaluators built on static code analysis, screenshot matching, or offline text comparison, the reward agent evaluates a *deployed and reachable* application, reproducing the user's usage process in a real browser. For each application it receives the original user query, the PRD, the application URL, and the task rubric. It layers four components—**Codex** as the core reasoning agent, an **LLM/VLM** for multimodal scoring, **Playwright** for browser control, and a real **Browser** runtime—decoupling high-level task understanding from low-level execution and evidence collection, so the system emits not only reward levels but the intermediate evidence behind them. The process has four stages: *task understanding* (determine the primary user behaviors and expected outcomes from query and PRD), *interaction verification* (autonomously plan test paths and perform navigation, input, clicks, submission, and result confirmation, adapting to intermediate states), *evidence aggregation* (task results, page states, interaction feedback, and multi-state screenshots), and *reward generation* (usability and aesthetics ratings, each with a rationale grounded in the evidence). Functionality is judged by whether a user can complete the target task through normal interaction—not by whether a component merely appears on the page. Rubrics constrain the judgment boundary for consistency and reviewability without replacing the agent's interaction and reasoning.

Aesthetics: a versioned nine-item checklist. Functional reward alone does not capture the overall experience, so AppWright introduces an *independent* aesthetics reward. Grounded in the same real-usage process, the agent observes the home page, key pages, and major interaction states on both desktop and mobile, saves the visual evidence, and applies a versioned nine-item checklist distilled from production design-review standards and user aesthetic feedback: layout hierarchy, content quality, color, typography, craftsmanship and finish, expressive aesthetics, cross-page consistency, global aesthetic quality, and positive visual strengths (full definitions in Appendix B). Each item is judged individually via multimodal scoring over multiple screenshots, then aggregated into a continuous score in $[0, 1]$ (the 47-item mean) and a binary pass/fail label. The rubric is versioned and iterated with the task suite, making aesthetics a reproducible, comparable evaluation axis rather than a holistic impression score: it does not equate "no visible visual error" with high quality, but rewards clear design intent, service to content comprehension, and cross-viewport consistency. The two axes are reported as two orthogonal rates over the full dataset—`usable_rate` $=$ #usable/#cases and `positive_rate` $=$ #positive/#cases—each produced by an independent judge policy with its own prompt, output schema, and cached manifest, so completing one does not satisfy the other; a case with no valid judge manifest counts as *incomplete* (an infrastructure, readiness, execution, or output-parsing failure) rather than a negative verdict. The aesthetic verdict uses the stricter of our two rubric policies, `robust_precision`: a page counts as `positive` only if it is evaluable, avoids every negative-dimension defect, *and* clears a positive-evidence threshold on the

Table 2: Agreement between the reward agent and human evaluation, measured on the outputs of a single model (Kimi K3, 184 applications, 10 annotators). Usability is validated in both directions; aesthetics is validated on the positive (high-aesthetic) class only, since designer consensus within the rejected class is low by construction (§5). The two human-ceiling rows bound what the agent precisions can be expected to approach: a model precision above the human ceiling is a sign of a labeling artefact rather than of skill. All rates are in %.

Metric	Value
Usability agreement	
Two-tier agreement (excellent / basically usable)	80.0
Three-tier agreement (excellent / basically usable / unusable)	60.0
Usability precision	
Excellent-tier precision (two-tier)	90.0
Excellent-tier precision (three-tier)	92.0
Aesthetics precision (positive class only)	
High-aesthetic precision (agent-labeled ∩ designer-labeled)	90.0
Human agreement (ceiling)	
Designer–designer agreement (high-aesthetic class)	90.0
Annotator–annotator agreement (usability, three-tier)	90.0

`visual_strengths` items, so a bug-free but visually plain page is not credited. Reporting the rates separately keeps "works but ugly" and "pretty but broken" distinguishable; the full accounting identities and the blank-application rule are in Appendix M.

Reliability. We validate the reward agent against human judgment on a deliberately narrowed subset, and state the narrowing rather than hide it. Annotating every model's output under two independent standards is not a tractable labeling budget: spreading the same effort across nine models would yield nine noisy estimates instead of one trustworthy one. We therefore annotate a single model, Kimi K3, the strongest system on the hardest tier (86.4% usable, field-best D4 excellent rate 40.9%, §6.2); validating against the strongest builder is the conservative direction, since its artifacts are the most complete and a positive result is informative rather than trivially satisfied. The annotation applies three standards—*usable*, *excellent*, and *aesthetically strong*—over the full Kimi K3 output set at 184 applications with 10 annotators trained on the production rubric. Usability holds up on both counts: two-tier agreement (excellent vs. basically usable) is 80.0% and three-tier agreement is 60.0%, with precision on the excellent verdict of 90.0% at two tiers and 92.0% at three. Aesthetics is validated one-sided by construction—designer consensus is high on the positive class and low within the rejected class—so we keep it as the binary gate of §5 and verify only the positive class, where precision is 90.0%. The full protocol is in Appendix L.

Coverage is traded for reproducibility. The task suite, scaffold, reward harness, and per-case reward outputs are all released, along with the agent verdicts and human labels of the reliability study, so every comparison in Table 2 is recomputable without repeating the annotation—which makes the single-model choice defensible, since the labels that validate the agent are public alongside its scores.

6 Experiments

6.1 Experimental Setup

All systems are evaluated under a single fixed reference scaffold running on our Code Arena harness reimplementation (§4), so score differences reflect the model rather than bespoke agent engineering. The scaffold is identical across models: a single OpenAI-compatible Chat Completions endpoint with `stream=true` and `parallel_tool_calls=false`; `MAX_TOOL_STEPS`=150 sequential tool-calling steps; and up to 4 model-request retries per rollout, with the full prior reasoning and tool history returned and never summarized. No rubric, difficulty label, acceptance checklist, or reference solution reaches the model; these are used only by the reward agent.

Table 3: Main results on APPWRIGHT, over the full 184-task denominator: a pair whose four rollouts all fail counts as *unusable* (aesthetic 0, label fail), not dropped. Rates in %; usable = excellent + basically usable. Aesthetic mean is the 47-item score with failures scored 0 (all-blank floor $36/47 = 76.60$); the pass rate uses the two-stage `robust_precision_v3` rule of §5, judging first-screen visual quality jointly with content, adaptation, and design consistency rather than thresholding the score. The four metrics have *three different leaders* (bold); rows are ordered by excellent rate, and no composite is reported, since fusing non-commensurable metrics would hide the divergence this benchmark isolates.

Model	Excellent	Usable	Aesth. mean	Aesth. pass
Qwen3.8-Max	**60.9**	88.6	84.54	37.5
Kimi K3	52.7	**91.3**	83.38	37.5
Claude Opus 4.8	48.9	83.2	77.73	13.0
HY4-Preview	47.8	90.8	77.31	14.7
Qwen3.8-Flash	47.8	89.1	**84.90**	**39.7**
DeepSeek-V4-Pro-0813	45.7	84.8	77.41	15.2
GLM-5.3	41.3	77.2	76.61	17.4
DeepSeek-V4-Flash-0731	37.5	78.3	81.09	24.5
Qwen3.8-27B	32.6	69.6	75.82	22.3
Mean	46.1	83.6	79.87	24.6

We evaluate nine frontier reasoning/coding models spanning open and proprietary families, including three capacity tiers within one current-generation family: GLM-5.3, DeepSeek-V4-Flash-0731, DeepSeek-V4-Pro-0813, Claude Opus 4.8, Kimi K3, HY4-Preview, Qwen3.8-27B, Qwen3.8-Flash, and Qwen3.8-Max. Each is invoked as the reasoning backend of the fixed scaffold with uniform decoding (reasoning effort `xhigh`, temperature 0.95, 1M-token context); no model receives task-specific prompt tuning (Appendix G).

All 184 cases are run for every model with four independent rollouts per model–task pair ($9 \times 184 = 1{,}656$ combinations), scored `pass@1` per case. Each application is scored by two *independent* judge policies with a pinned backbone (Codex reasoning agent + LLM/VLM scoring + Playwright/Chromium), run serially as *usability* → *teardown* → *aesthetic* from independent restores; the judge model for all reported runs is `gpt-5.5`. We report four orthogonal figures over the full dataset: the *excellent rate* and *usable rate* (usable = excellent + basically usable), and the *aesthetic mean* (47-item score, failures scored 0) and *aesthetic pass rate*. *Failures are never dropped*: a case whose four rollouts all fail is recorded as *unusable* with aesthetic score 0 and label fail—deliberately, since excluding generation failures would flatter every model and hide the failure mode separating the top and bottom of the field.

6.2 RESULTS

Table 3 reports the four quality metrics for all nine models (per-category breakdown in Appendix D); detailed decomposition is deferred to Appendix H. Three findings are the substance of the evaluation.

Finding 1: no single metric identifies the best system. Qwen3.8-Max posts the highest excellent rate (60.9%), Kimi K3 the highest usable rate (91.3%), and Qwen3.8-Flash the highest aesthetic mean and pass rate (84.90, 39.7%)—three different leaders across four metrics. These are not interchangeable orderings of one quality: Qwen3.8-Max delivers 112 excellent, 51 basically-usable, and 21 unusable cases, whereas Kimi K3 delivers 97, 71, and 16, so Max wins by 15 excellent cases while Kimi wins by 5 unusable cases: excellent measures how often a task is met to full standard, usable how rarely it fails outright. Aesthetics dissociates most sharply: Claude Opus 4.8 holds the third-highest excellent rate (48.9%) but the *lowest* aesthetic pass rate (13.0%), and its aesthetic *mean* (77.73) is not the lowest, so its deficit is the binary gate rather than accumulated defect penalties. Capability here is a profile across four metrics, not a position: the open-weight Qwen3.8 family leads the excellent rate and both aesthetic figures, ahead of several top-tier closed models, so a model selected on one axis would be misjudged on the other.

Table 4: Difficulty sets the ceiling: metrics pooled across all nine models, by difficulty tier. Rates in %; usable = excellent + basically usable. The cross-model spread is the range of per-model usable rates within the tier.

Metric	D1	D2	D3	D4
Tasks	47	52	63	22
Usable	91.3	91.9	79.0	61.1
Excellent	53.0	55.1	41.3	24.2
Aesthetic mean	81.8	81.5	79.8	71.9
Aesthetic pass	24.3	30.1	25.2	10.6
Cross-model spread	8.5	–	–	45.5

Finding 2: difficulty, not application category, sets the ceiling. Difficulty is the dominant axis. Table 4 pools all nine models by tier: usability and excellence fall monotonically, with a cliff at D4 (usable $91.3 \rightarrow 91.9 \rightarrow 79.0 \rightarrow 61.1\%$) and the same shape on both aesthetic figures. The tiers also differ in what they separate: the cross-model spread in usable rate is only 8.5 points at D1 but 45.5 at D4, so the easy tiers rank no one and the hardest one decides the leaderboard. Kimi K3 is the most robust on D4 (86.4% usable, field-best D4 excellent rate 40.9%), which lifts it to the top of overall usability, whereas Qwen3.8-Max wins by breadth—leading excellent rate on D1–D3 but falling to 36.4% at D4. Category, by contrast, barely moves usability: pooled rates span only 82.1–85.5% across the five categories, against 30.8 points across tiers (per-category table in Appendix F).

Finding 3: aesthetics must be reported as a gate, not a mean. The 47-item checklist mean spans only 75.82–84.90 across the field—a 9-point range on a 0–100 scale—while the binary pass rate spans 13.0–39.7%, a factor of three, and even the best model fails the gate on over 60% of tasks. The scoring structure explains the compression: with 36 negative and 11 positive items, a page with no positives but no negatives already scores 76.60, close to where the field sits. Discrimination comes almost entirely from the single positive dimension `visual_strengths` (55.6% of the summed weighted discrimination across the nine dimensions, over four times the next-largest contributor), while the negative dimensions saturate (color 93.9–98.9, typography 97.8–99.2, cross-page consistency 96.6–99.1): aesthetic rank reflects whether a model produces deliberate visual highlights, not whether it avoids errors (Appendix H).

7 Discussion

The reward agent reproduces what a production application-generation service uses to judge its own output: usable, excellent, and the aesthetic gate are the axes a deployed no-code service applies before shipping, so AppWright measures models against the standard they are held to. An automated judge is viable because the task resists exact evaluation—no reference to diff against, subjective criteria, and human judgment that does not scale to nine models over 184 tasks—and the reliability study (§5) licenses the substitution: at 80.0% two-tier human agreement and 90.0% excellent-verdict precision, the agent screens as a human panel would, which is how production uses it alongside a sampled human pass.[2]

8 Conclusion

AppWright evaluates application generation the way a production service does: one fixed harness, a five-category suite, and a dual-axis reward agent judging whether a delivered application works and whether it is presentable, as two independent rates rather than one aggregate. The axes disagree—the best model differs by metric, difficulty rather than category decides the ranking, aesthetics discriminates only as a gate—exactly what a single aggregate would hide. We release the suite, harness, agent, and labels so every number is recomputable.

References

AfterQuery. App-bench: Evaluating ai app builders on full-stack applications. `https://appbench.ai/`, 2025.

[2]Reproducible without human cost; Appendix N takes up the four limits.

Arena. Code arena: Web development leaderboard. https://arena.ai/leaderboard/code/webdev, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

John Brooke. Sus: A "quick and dirty" usability scale. Technical report, Usability Evaluation in Industry, 1996.

ByteDance Web-Bench Team. Web-bench: A sequential project-level benchmark for web development. *arXiv preprint arXiv:2505.07473*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cookie-Bench Authors. Cookie-bench: A web development benchmark with multi-stage usability and aesthetics verification. *arXiv preprint arXiv:2605.30000*, 2026.

Design Arena. Design arena: A live leaderboard for ai design quality. https://www.designarena.ai/about, 2025.

DesignBench Authors. Designbench: Evaluating multimodal llms on ui generation, editing, and repair. *arXiv preprint arXiv:2506.06251*, 2025.

FrontendBench Authors. Frontendbench: Evaluating llms on realistic front-end functional tasks. *arXiv preprint arXiv:2506.13832*, 2025.

FullStack-Bench Authors. Fullstack-bench: Evaluating coding agents on complex full-stack website construction. *arXiv preprint arXiv:2602.03798*, 2026. ICML 2026.

Google and others. From prompt to product: A user study of ai app-building platforms. *arXiv preprint arXiv:2512.18080*, 2025.

InteractWeb-Bench Authors. Interactweb-bench: Persona-driven dynamic web application construction. *arXiv preprint arXiv:2604.27419*, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

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

Jung et al. Ui-bench: Expert pairwise evaluation of text-to-app design quality. *arXiv preprint arXiv:2508.20410*, 2025.

Bettina Laugwitz, Theo Held, and Martin Schrepp. Construction and evaluation of a user experience questionnaire. *HCI and Usability for Education and Work*, pp. 63–76, 2008.

Li et al. Webdevjudge: Validating automated judges for web development. In *International Conference on Learning Representations (ICLR)*, 2026.

Liu et al. Productwebgen: Generating product showcase web pages with consistent visual assets. *arXiv preprint arXiv:2606.01022*, 2026a. KDD 2026.

Liu et al. Webcoderbench: A multi-metric benchmark for real-world web application generation. *arXiv preprint arXiv:2601.02430*, 2026b. ACL 2026.

LMArena. Webdev arena: A live llm leaderboard for web app development. https://web.lmarena.ai, 2025.

Lu et al. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch. *arXiv preprint arXiv:2505.03733*, 2025. NeurIPS 2025 Datasets and Benchmarks Track.

MiniAppBench Authors. Miniappbench: Evaluating application generation for mini-program runtimes. *arXiv preprint arXiv:2603.09652*, 2026. ICML 2026 Spotlight.

Morten Moshagen and Meinald T Thielsch. Facets of visual aesthetics. *International Journal of Human-Computer Studies*, 68(10):689–709, 2010.

Morten Moshagen and Meinald T Thielsch. A short version of the visual aesthetics of websites inventory. *Behaviour & Information Technology*, 32(12):1305–1311, 2013.

PlayCoder Authors. Playcoder: Making llm-generated gui code playable. *arXiv preprint arXiv:2604.19742*, 2026.

Revelo. Ui quality bench: A designer-rated benchmark of generated interfaces. `https://huggingface.co/datasets/revelohq/ui-quality-bench`, 2026.

Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are we from automating front-end engineering? In *NAACL*, 2025.

Tencent WorkBuddy Team. Workbuddy bench: Evaluating coding agents on front-end and gui work activities. *arXiv preprint arXiv:2607.20911*, 2026.

Terminal-Bench Team. Terminal-bench: Benchmarking agents in the terminal. `https://github.com/laude-institute/terminal-bench`, 2025.

Vision2Web Authors. Vision2web: Benchmarking coding agents on prototype-to-website generation. *arXiv preprint arXiv:2603.26648*, 2026. ICML 2026 Spotlight.

VISTA Authors. Vista: Anchor-based evaluation of ui code generation from multi-fidelity inputs. *arXiv preprint arXiv:2605.26144*, 2026.

W3C. Web content accessibility guidelines (wcag) 2.2. `https://www.w3.org/TR/WCAG22/`, 2023.

Wang et al. Webgen-v bench: Section-level visual implementation quality for open-ended web generation. *arXiv preprint arXiv:2510.15306*, 2026. KDD 2026.

WebCompass Authors. Webcompass: A multimodal benchmark for web generation, editing, and repair. *arXiv preprint arXiv:2604.18224*, 2026.

Wu et al. Uiclip: A data-driven model for assessing user interface design. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2024.

Xiao et al. Code aesthetics with agentic reward feedback. *arXiv preprint arXiv:2510.23272*, 2026. OpenDesign / AesCoder; ICLR 2026.

John Yang, Carlos E Jimenez, et al. Swe-bench multimodal: Do ai systems generalize to visual software domains? *arXiv preprint arXiv:2410.03859*, 2024.

Zhang et al. Artifactsbench: Bridging the visual-interactive gap in llm code generation evaluation. *arXiv preprint arXiv:2507.04952*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Terry Yue Zhuo et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

REPRODUCIBILITY STATEMENT

All artifacts needed to reproduce every number in this paper are released. *Code.* The open-source repository (`https://github.com/Azure99/app_wright`) contains the construction-and-execution harness and the reward-agent harness: the rollout worker, the tool schemas and template registry, the container images and pinned runtimes, the usability and aesthetic judge policies with their prompts and output schemas, and the aggregation and gating code. The reward agent runs the Codex harness with `gpt-5.5` at `xhigh` reasoning effort, so its verdicts are reproducible from the released code and rubrics alone, at no human cost. *Data.* The task suite—184 tasks with their queries, structured PRDs, rubrics, and difficulty labels—together with the generated applications and the per-case reward and human-annotation outputs is hosted at `https://huggingface.co/datasets/dongdaxiang/app_wright`. *Pinning.* Model endpoints are served through OpenRouter with the served versions and decoding parameters recorded per release; the judge backbone, the benchmark/rubric version, and the container image tag are pinned, so a rerun reproduces the reported figures exactly. *Reliability.* Because the agent verdicts and the human labels are released on the same applications, every agreement figure in §5 and Table 2 can be recomputed without repeating the annotation.

A TASK EXAMPLES

The released suite documents each task in full: the desensitized requirement text, the structured PRD with its acceptance criteria, the difficulty label D1–D4, the implementation-form tag, and the per-axis rubrics the reward agent applies. One worked example per category—web app, tool, game, mini program, and mobile app—is included in the dataset release alongside the complete task files, so the construction conventions of §3 can be inspected directly rather than inferred.

B AESTHETICS RUBRIC (FULL VERSION)

The aesthetics reward is a nine-item checklist applied individually to each observation state, with per-item definitions, anchors, and good/bad exemplar pairs; the versioned rubric is released with the dataset. Each item is scored over the same multi-state screenshots, and the item scores aggregate into the 47-item mean and the two-stage binary gate of §5. The dimensions are `layout_hierarchy` (4 items), `content_quality` (5), `color_quality` (4), `typography_craft` (4), `craftsmanship_finish` (5), `expressive_aesthetics` (9), `cross_page_consistency` (3), `global_aesthetic_quality` (2), and `visual_strengths` (11, the sole positive dimension). The first eight comprise 36 negative items, each asking only that a page avoid a named defect; a page with no positives and no negatives scores $36/47 \times 100 = 76.60$, the structural reference value discussed in §H.1. Under the reported `robust_precision` policy a page is `positive` only if it is evaluable, avoids every negative-dimension defect, and clears the positive-evidence threshold on `visual_strengths`.

C HARNESS DETAILS

This appendix carries the engineering specification whose body text was condensed in §4.

Rollout lifecycle and success commit. Each case runs in a fresh worker attempt inside an isolated container, with nothing carried over from other cases. The lifecycle is: read and validate the case (`case_id`, a fixed `template_id`, the query, and the PRD); check for an existing valid rollout slot and, on a miss, create a unique attempt; copy the fixed *starter workspace* and render the template's system prompt; then enter the agent loop. The loop issues a single Chat Completions request; if the model returns tool calls it executes them, appends the `assistant` and `tool` messages, and issues the next request; the loop ends only when a completion returns `stop` with no pending tool call. The worker then runs install/build, launches the app for a readiness check, seals the verified `source`/`runtime`, and performs the atomic `manifest.json` write that is the sole success-commit point. Here "cache" denotes a local sealed-artifact slot (a completed rollout that can be reused), not model-output caching.

Template prompt and tool set. The first turn contains exactly two messages: a `system` message rendering the template's prompt and a `user` message concatenating the query with the PRD (*"...\n\nProduct requirements:\n"*). The template prompt fixes the target stack and the budget: the `react-vite-tailwind` template instructs the model to build a website using tools only (never emitting code in prose), reserves `App` in `src/App.tsx` as the entry point, forbids editing `package.json`/`vite.config.ts` directly, and caps the run at `MAX_TOOL_STEPS` sequential tool-calling steps (150 in the released template). Streamed argument deltas are merged before dispatch; we distinguish a *tool step* (one executed tool call) from a *completion* (one model turn), and HTTP retries do not increase the completion count.

The `react-vite-tailwind` template exposes 13 tools as JSON-Schema function definitions: file-workspace operations (`read_file`, `create_file`, `edit_file`, `delete_file`, `list_files`, `grep_files`, `glob_files`), dependency and validation (`install_npm_packages`, `build_project`), and external research/media (`web_search`, `web_fetch`, `fetch_media`, `generate_image`).

Isolation and redaction. Isolation is enforced at the tool boundary: file paths must be workspace-relative, and out-of-bounds paths, excluded directories, symlinks, and ill-formed arguments are rejected. Tool results pass through secret redaction before reaching the model. The model never sees provider credentials, host absolute paths, the Docker socket, other cases, or unauthorized tools.

Runtimes and trace format. Aesthetic verification observes fixed viewports (`1280x800` and `390x844`) plus key interaction states. Every run logs build logs, runtime logs, multi-state screenshots, and the interaction trace, which serve both the reward agent and human audit. The full engineering specification—container images and pinned runtime versions, the agent interface schema, mini-program simulator setup, the trace format, and the exact released image tag with a one-command reproduction entry point—ships in the released repository alongside the harness code named in the reproducibility statement.

D Full Results

Table 3 reports the four headline quality metrics (excellent, usable, aesthetic mean, aesthetic pass) for all nine models; this appendix records the decompositions behind them. The per-model difficulty×category matrices are in Appendix E, the pooled per-category usability breakdown in Appendix F, the per-dimension aesthetic decomposition in Appendix H, and the per-model inference configuration in Appendix G. Per-difficulty and per-dimension profiles beyond these are recoverable from the released per-case reward outputs, which carry the full metric vector for every application in the suite.

E Per-Model Difficulty×Category Matrices

Each cell is the pair `usable`/`positive` (%) for that model at a given difficulty (rows D1–D4) and category (columns Web, Tool, Game, Mobi, Mini); the left number traces usability, the right aesthetics. Small cells (Mobile App n=12, D4 n=22) are high-variance and shown for completeness. See §6.2 for the reading.

F Per-Category Usability

Usability (%) by application category, rows ordered by 3. Pooled category means span only 3.4 points, against 30.8 across difficulty tiers (§6.2).

G Per-Model Inference Configuration

Harness-level settings (`MAX_TOOL_STEPS`=150, four rollouts per case, `stream=true`, `tool_choice=auto`, `parallel_tool_calls=false`) are shared across all models; end-

Kimi K3

	Web	Tool	Game	Mobi	Mini
D1	100/43	84/21	100/36	100/0	100/62
D2	93/40	100/40	100/73	100/40	92/46
D3	94/44	88/12	92/54	100/67	67/25
D4	90/0	100/0	60/50	100/0	100/33

HY4-Preview

	Web	Tool	Game	Mobi	Mini
D1	100/14	100/5	91/0	67/33	88/0
D2	93/20	90/0	91/9	100/0	100/23
D3	100/19	62/12	92/33	100/33	100/17
D4	70/10	67/0	80/20	0/0	100/0

Qwen3.8-Flash

	Web	Tool	Game	Mobi	Mini
D1	100/43	79/53	100/45	100/67	100/25
D2	100/53	100/50	91/64	100/20	100/38
D3	94/19	75/0	83/75	67/0	100/17
D4	50/10	100/33	80/20	0/0	67/0

Qwen3.8-Max

	Web	Tool	Game	Mobi	Mini
D1	100/43	95/37	91/55	67/67	88/25
D2	100/40	90/40	100/27	100/20	92/62
D3	88/27	88/12	83/54	100/67	75/33
D4	80/10	67/33	80/20	0/0	100/0

DSV4-Pro

	Web	Tool	Game	Mobi	Mini
D1	100/14	89/5	100/36	100/33	100/12
D2	100/27	100/20	91/0	100/20	100/15
D3	75/12	50/0	83/25	100/0	83/8
D4	40/10	67/33	60/25	0/0	67/0

Opus 4.8

	Web	Tool	Game	Mobi	Mini
D1	86/14	95/16	100/18	100/0	75/25
D2	93/13	70/10	82/36	100/20	100/8
D3	88/6	62/0	75/21	100/0	58/0
D4	90/0	67/33	60/0	0/0	67/0

DSV4-Flash

	Web	Tool	Game	Mobi	Mini
D1	86/0	89/5	82/64	67/0	100/25
D2	93/33	80/10	82/36	80/20	92/38
D3	81/38	62/0	71/42	100/0	83/8
D4	30/10	100/33	40/40	0/0	33/0

GLM-5.3

	Web	Tool	Game	Mobi	Mini
D1	86/29	89/11	100/27	67/0	88/25
D2	93/27	90/20	73/36	100/0	92/23
D3	94/12	71/0	62/25	33/0	75/9
D4	40/0	33/0	40/20	0/0	67/0

Qwen3.8-27B

	Web	Tool	Game	Mobi	Mini
D1	71/14	95/17	91/45	100/0	75/12
D2	87/27	90/30	91/73	60/20	46/8
D3	62/38	25/14	62/21	67/33	75/8
D4	30/11	100/0	40/0	0/0	67/0

Table 5: Per-model joint *usability/aesthetics* matrix: each cell is `usable`/`positive` pass rate for that model's evaluated cases at a given difficulty (rows D1–D4) and application category (columns Web, Tool, Game, Mobi=Mobile App, Mini=Mini Program); all values are in % and a "–" marks a category with no evaluated case for that model. Usability uses the `usable` verdict; aesthetics uses the stricter two-stage rule of §5. Small cells (Mobile App n=12, D4 n=22) are high-variance and shown for completeness: one case moves a D4 cell by 4.5 points and a Mobile-App cell by 8.3.

Table 6: Usability by application category (values in %). Category sizes: Game 49, Web 47, Tool 40 (including the five H5 tasks), Mini Program 36, Mobile App 12; the small Mobile-App split (n=12) is reported for completeness but is high-variance, since a single case moves the cell by 8.3 points. Rows follow the overall ordering of Table 3. The spread across categories within a model (pooled means 82.1–85.5%, a 3.4-point range) is far smaller than the spread across models within a category, and smaller still than the 30.8-point spread across difficulty tiers—category is a secondary factor relative to difficulty (§6.2).

Model	**Game**	**Web**	**Tool**	**Mini Prog.**	**Mobile App**
Qwen3.8-Max	88	92	90	86	83
Kimi K3	92	94	90	86	100
Claude Opus 4.8	82	89	80	78	92
HY4-Preview	90	92	88	97	83
Qwen3.8-Flash	90	87	85	97	83
DeepSeek-V4-Pro-0813	86	79	83	92	92
GLM-5.3	69	81	80	83	67
DeepSeek-V4-Flash-0731	71	77	83	86	75
Qwen3.8-27B	71	64	80	64	67
Mean	82.1	83.7	84.2	85.5	82.4

points are served through OpenRouter. Decoding is also uniform: reasoning effort `xhigh`, temperature 0.95, and a 1M-token context window for every model.

H Analysis Details

This appendix carries the detailed analysis whose headline findings are stated in §6.2.

Table 7: Per-model inference configuration. Harness-level settings (MAX_TOOL_STEPS=150, four rollouts per case, stream=true, tool_choice=auto, parallel_tool_calls=false) and decoding settings (reasoning effort xhigh, temperature 0.95, 1M-token context) are shared across all models.

Model	Endpoint ID	Reasoning effort	Temp.	Context
GLM-5.3	–	xhigh	0.95	1M
DeepSeek-V4-Flash-0731	ds/deepseek-v4-flash-0731	xhigh	0.95	1M
DeepSeek-V4-Pro-0813	ds/deepseek-v4-pro-0813	xhigh	0.95	1M
Claude Opus 4.8	–	xhigh	0.95	1M
Kimi K3	–	xhigh	0.95	1M
HY4-Preview	–	xhigh	0.95	1M
Qwen3.8-27B	–	xhigh	0.95	1M
Qwen3.8-Flash	–	xhigh	0.95	1M
Qwen3.8-Max	–	xhigh	0.95	1M

H.1 Aesthetics Concentration and Dimension Decomposition

The aesthetics axis behaves very differently from the usability axis: it *concentrates at the top while the binary gate stays closed*. The 47-item checklist mean spans only 75.82 to 84.90 across the whole field—on a 0–100 scale, a total range of 9 points—while the aesthetic pass rate spans 13.0 to 39.7%, a factor of three. The scoring structure explains the compression. The checklist contains 36 negative items (each asking only that a page avoid a named defect) and 11 positive items (each asking that a visual highlight be present); a page with no positives but no negatives still scores $36/47 \times 100 = 76.60$. This is a structural reference value rather than a lower bound—negative defects subtract and positive highlights add—but it is close to where the field actually sits, and it is why the mean cannot rank models. The negative dimensions are effectively saturated: pooled across models, color quality sits at 93.9–98.9, typography at 97.8–99.2, and cross-page consistency at 96.6–99.1, leaving almost no room for between-model separation. Indeed the highest aesthetic mean in Table 3 (84.90) exceeds the no-positives reference by only 8.3 points, so the entire positive contribution to the field's best score is under a tenth of the scale.

The between-model discrimination comes almost entirely from the single positive dimension, visual_strengths. Table 8 reports each dimension's weighted discrimination—its cross-model mean range multiplied by its share of the 47 items—and the asymmetry is stark: visual_strengths contributes 6.30 points, or 55.6% of the summed weighted discrimination across all nine dimensions, more than four times the next-largest contributor, expressive_aesthetics (1.54 points, 13.6%). Every remaining dimension contributes under 1.2 points, and typography and cross-page consistency together contribute 0.28 points—less than 5%—despite comprising 7 of the 47 items. The mechanism is visible in the raw ranges: the high-mean band scores 39.8–54.7 on visual_strengths while the low band scores only 27.8–32.4, with no overlap, whereas their negative-dimension scores are nearly indistinguishable. A model's aesthetic rank therefore reflects whether it produces deliberate visual highlights, not whether it avoids errors. As one concrete comparison, Qwen3.8-Flash and Claude Opus 4.8 differ by 7.17 points in total aesthetic mean, of which roughly 6.3 points is attributable to the visual_strengths dimension alone ($(54.7 - 27.8) \times 11/47$); Opus is in fact *higher* than Qwen3.8-Flash on color quality, cross-page consistency, and global aesthetic quality, and those advantages are still not enough to close the gap.

One caveat on this grouping: the two clusters (81.09–84.90 and 75.82–77.73, separated by 3.36 points against a 1.91-point span within the lower band) describe the distribution of *model-average* scores. It does not establish that per-task scores are bimodal, and we do not claim they are.

Two design implications follow. For the benchmark, the near-saturated negative dimensions contribute little signal at the current capability level and should either be tightened or reweighted; the mean score is a weak ranking instrument and the binary gate is the figure to trust—consistent with reporting the stricter two-stage rule as primary. For models and training, the gap is not "can the model render a correct layout" (they can) but "does it exercise visual intent"—the highest-leverage behaviors are the expressive ones the positive items isolate. Aesthetics also decays with difficulty in the same direction as usability, but it is governed far more by application type: pooled across

Table 8: Weighted discrimination by aesthetic dimension. *Weighted discrimination* is the cross-model range of the dimension mean multiplied by the dimension's share of the 47 items; the final column expresses each dimension's contribution as a share of the summed weighted discrimination across all nine dimensions (total 11.33 points). Computed from unrounded dimension means.

Dimension	Items	Mean range	Share
`visual_strengths` (pos.)	11	27.8–54.7	**55.6%**
`expressive_aesthetics`	9	86.5–94.6	13.6%
`content_quality`	5	86.7–97.6	10.2%
`layout_hierarchy`	4	80.7–88.9	6.1%
`global_aesthetic_quality`	2	79.1–90.8	4.4%
`craftsmanship_finish`	5	91.3–95.4	3.9%
`color_quality`	4	93.9–98.9	3.8%
`cross_page_consistency`	3	96.6–99.1	1.4%
`typography_craft`	4	97.8–99.2	1.0%

models, Game is the aesthetic-friendliest category (84.0 mean, 36.5% pass) and Tool the least (78.8, 17.5%), a 19-point spread in pass rate against the 3.4-point spread for usability. Notably, all nine models achieve their own best category aesthetic score on Game, so the effect is not driven by one model.

Joint difficulty×category structure. Appendix E exposes every model as a 4×5 difficulty×category grid of `usable`/`positive` rates (Table 5). Three regularities reinforce §6.2: usability decays with difficulty and the spread across models widens from 8.5 points at D1 to 45.5 at D4, with every model's weakest cell in D3–D4; category moves usability only marginally but moves aesthetics strongly (Game 36.5% pooled pass vs. Tool 17.5%); and the two numbers are uncorrelated cell by cell, so a model strong on one axis is not reliably strong on the other within the same stratum. The aesthetics head is also cell-sparse—only the leading models clear 30% outside Game, and the trailing group sits near the floor by D3–D4, the matrix-level signature of the concentration in §H.1.

H.2 Failure-Mode Attribution

The metric divergence of §6.2 is only interpretable if the underlying failure modes are. We attribute the aesthetic gate directly, since the perplexing case is a model that builds well and still fails the gate. Claude Opus 4.8 is the sharpest instance—third on excellent rate (48.9%), last on aesthetic pass (13.0%), and seventh of nine on aesthetic mean (77.73). We compared it against other models on four identical tasks, scoring each on the 11 positive checklist items.

The dominant failure mode is *structurally complete but thematically generic*. On a weather-personality quiz (miaoda_008), Opus hits 1/11 positive items: the page is an input field, text, and a soft background, so the weather theme never reaches the layout. On a learning-and-development department site (miaoda_014), also 1/11—the dark grid and card framework is complete, and the connection to the business is carried entirely by copy; substituting the department name would leave a generic corporate template. Here GLM-5.3 scores 0/11 *with* topical photography on the page, which shows that including imagery is not the same as establishing a visual motif. On a student-achievement showcase (miaoda_037) Opus reaches 4/11—rounded cards, stat blocks, and emoji—but still misses a high-quality visual subject, whereas Kimi K3 (9/11) and DeepSeek-V4-Flash-0731 (8/11) build the same information into a space-cadet/rocket motif and an illustrated growth timeline respectively. Accumulating local decoration, in other words, does not satisfy a gate that jointly requires a key visual subject and complete delivery.

The second mode is *delivery truncation*, and it is distinct from aesthetics proper. On a drone-fleet management system (miaoda_019), Opus's login and home pages carry a flight-control theme, but the device page fails to load, so the visual experience does not survive into the core business flow (2/11). This is a genuine incompleteness in the delivered artifact, not a conservative style choice—entrance-page polish does not guarantee that the rest of the application renders at all.

Two quantitative points follow. Across the four cases Opus's positive hits are 1, 1, 2, and 4 out of 11, and it misses the high-quality-visual-subject item in every one; the passing models on the same tasks

Table 9: Positioning against representative recent benchmarks. "App forms" counts distinct application categories covered; "Live run" means the evaluated artifact is built, served, and exercised rather than inspected statically; "Struct. rubric" means acceptance is a checkable, per-task criteria list; and the last two columns distinguish whether visual quality is scored at all and whether it is reported as a *separate* judged rate rather than folded into a preference or composite label. Only the design-quality instruments are non-competitive here (they measure artifacts, not generations); they are included to show where the construct originates.

Benchmark	**App forms**	**Live**	**Rubric**	**Visual**	**Sep. axis**
SWE-bench (Jimenez et al., 2024)	1	✓	✓	–	–
Design2Code (Si et al., 2025)	1	–	–	similarity	–
WebGen-Bench (Lu et al., 2025)	1	✓	✓	–	–
Vision2Web (Vision2Web Authors, 2026)	1	✓	✓	partial	–
WebCoderBench (Liu et al., 2026b)	1	✓	✓	partial	–
MiniAppBench (MiniAppBench Authors, 2026)	1	✓	✓	–	–
Cookie-Bench (Cookie-Bench Authors, 2026)	1	✓	✓	✓	–
ArtifactsBench (Zhang et al., 2025)	1	✓	–	✓	–
Code Arena (Arena, 2024)	1	✓	–	preference	–
WebDev Arena (LMArena, 2025)	1	✓	–	preference	–
UI Quality Bench (Revelo, 2026)	–	–	✓	✓	n/a
UIClip (Wu et al., 2024)	–	–	–	✓	n/a
APPWRIGHT (ours)	**5**	✓	✓	✓	✓

reach 5–10. But the hit count alone does not decide the gate: DeepSeek-V4-Pro-0813 reaches $5/11$ on miaoda_008 and still fails, confirming that pass/fail is a joint requirement over first-screen quality, adaptation, consistency, and the positive dimensions rather than a tally. These four cases illustrate a mechanism behind the field's lowest pass rate; they are an existence proof, not an estimate of how often each mode occurs across the suite, and the overall ranking remains determined by the full-set statistics.

H.3 REWARD-AGENT ABLATIONS

The headline agreement figures compare the released reward agent against human labels; a natural question is which components of the agent are load-bearing. The agent's design couples three choices—active interaction with the deployed application rather than screenshots alone, a per-task structural rubric rather than holistic judgment, and two independently policy-scored axes rather than one composite. The reliability study of §5 and Table 2 is run with all three in place; we report the full configuration as the evaluated system and leave a component-wise ablation—each choice removed in turn, scored against the same human labels, plus sensitivity to the judge backbone—to the released harness, whose policies are swappable.

I POSITIONING AND SUITE COMPOSITION

J CONSTRUCTION PIPELINE AND QUALITY CONTROL (DETAIL)

DEDUPLICATION AND PRD PREPARATION

We first deduplicate the raw query pool by semantic similarity between query embeddings, *before* PRD generation and difficulty annotation, so that near-duplicate requests cannot inflate any category or difficulty bucket before balancing. User queries and their materials are then sanitized, after which a fixed-instruction LLM agent generates a structured PRD organized around goals, page/module structure, core functionality, business rules, boundary conditions, and acceptance criteria; application types carry type-specific duties (web apps must state page hierarchies and cross-page rules; tools explicit inputs, outputs, and processing rules; games explicit mechanics and termination conditions), and acceptance criteria are written as a concise sequence of user actions, which is what makes the PRD checkable rather than merely descriptive. The generation prompt enforces three

Table 10: APPWRIGHT composition: joint distribution of 184 query–PRD tasks over five application categories (rows) and four difficulty levels (columns). Row and column marginals are exact.

Category	D1	D2	D3	D4	#Tasks	Share
Game	10	10	24	5	49	26.6%
Web app	7	14	16	10	47	25.5%
Tool	19	10	8	3	40	21.7%
Mini program	8	13	12	3	36	19.6%
Mobile app	3	5	3	1	12	6.5%
#Tasks	47	52	63	22	184	100%
Share	25.5%	28.3%	34.2%	12.0%	100%	

controls. *Intent preservation*: the agent retains the user's explicit requirements and does not alter the core goal. *Minimum scope*: it specifies only what that goal needs, admitting supporting capabilities (e.g. persistence, where records must remain accessible) only when the intended workflow cannot complete without them; unrelated enhancements are excluded and out-of-scope functionality stated explicitly. *No clarification round-trip*: the rewriting stage does not solicit clarification, and default choices are restricted to completing the core workflow. Specifications stay technology-agnostic, leaving framework choices to construction.

BALANCING AND COMPATIBILITY FILTERING

Each task is then assigned a difficulty level using the PRD-based procedure of §3.1, after which we downsample the annotated pool to balance coverage across categories and difficulty levels. The final per-category counts are not fixed in advance; they emerge jointly from this sampling and from compatibility filtering, so balancing means comparable coverage, not equal counts. We exclude tasks whose requirements depend on capabilities the harness or reward environment cannot support, since these would be scored on artefacts neither side can exercise: construction-side exclusions cover external service dependencies (AI services, payment, translation, phone-number or WeChat authentication, SMS, e-commerce, map APIs), and evaluation-side exclusions cover speech synthesis and recognition, audio playback, camera access, and system-level controls the reward environment cannot adequately operate or verify.

CONTAMINATION CONTROL

Because the tasks are drawn from a production service rather than public corpora, none of the evaluated material has been publicly available at any point, and the ordinary contamination route—a model having seen the evaluation set during training—does not apply. Three properties make this concrete. *(1) A post-training window.* The queries come from traffic collected between June and September 2026, a period after the release of every model evaluated here, so the evaluation text could not have been in any model's training data. *(2) Never public.* The source traffic is proprietary and is not published in any form, so the released suite is not reproducible from any public source. *(3) Paraphrased and desensitized.* Every exposed query passes a desensitization review and is then *paraphrased under a meaning-preservation constraint*, so the released text is neither the verbatim user input nor a scraped artifact, while its semantics are held fixed; the accompanying PRD is machine-generated from that query by a fixed template, not retrieved from any document. The suite is therefore a non-contaminated sample: new text, drawn from a period after the models' training, from a source that has never been public.

K APPLICATION GENERATION HARNESS (DETAIL)

ROLLOUT LIFECYCLE.

Each case runs in a fresh worker attempt inside an isolated container, with nothing carried over from other cases. The lifecycle is: read and validate the case (`case_id`, a fixed `template_id`, the

query, and the PRD); check for an existing valid rollout slot and, on a miss, create a unique attempt; copy the fixed *starter workspace* and render the template's system prompt; then enter the agent loop. The loop issues a single Chat Completions request; if the model returns tool calls it executes them, appends the `assistant` and `tool` messages, and issues the next request; the loop ends only when a completion returns `stop` with no pending tool call. The worker then runs install/build, launches the app for a readiness check, seals the verified `source`/`runtime`, and performs the atomic `manifest.json` write that is the sole success-commit point.

TEMPLATE PROMPT AND TOOL SET.

The first turn contains exactly two messages: a `system` message rendering the template's prompt and a `user` message concatenating the query with the PRD. The template prompt fixes the target stack and the budget: the `react-vite-tailwind` template instructs the model to build a website using tools only (never emitting code in prose), reserves `App` in `src/App.tsx` as the entry point, forbids editing `package.json`/`vite.config.ts` directly, and caps the run at `MAX_TOOL_STEPS` sequential tool-calling steps (150 in the released template). We distinguish a *tool step* (one executed tool call) from a *completion* (one model turn), and HTTP retries do not increase the completion count.

ISOLATION, REDACTION, AND RUNTIMES.

Isolation is enforced at the tool boundary: file paths must be workspace-relative, and out-of-bounds paths, excluded directories, symlinks, and ill-formed arguments are rejected. Tool results pass through secret redaction before reaching the model, so the model never sees provider credentials, host absolute paths, the Docker socket, other cases, or unauthorized tools. Aesthetic verification observes fixed viewports (`1280x800` and `390x844`) plus key interaction states. Every run logs build logs, runtime logs, multi-state screenshots, and the interaction trace, which serve both the reward agent and human audit.

L RELIABILITY PROTOCOL (DETAIL)

WHY A SINGLE ANNOTATED MODEL.

Spreading the same labeling effort across nine models would yield nine noisy estimates instead of one trustworthy one, so we annotate the full output set of Kimi K3 alone. Kimi K3 is the strongest system in our evaluation on the hardest tier (86.4% usable and the field's best D4 excellent rate at 40.9%, §6.2). The direction of the validation is conservative: the strongest builder produces the most complete artifacts, so agreement there is not rescued by obvious failure, and a positive result is informative rather than trivially satisfied. The annotation applies three standards to the same applications—is the delivered application *usable*, is it *excellent*, and is it *aesthetically strong*—at 184 applications with 10 annotators trained on the production rubric. Usability agreement is 80.0% at two tiers (excellent / basically usable) and 60.0% at three tiers (excellent / basically usable / unusable); precision on the excellent verdict is 90.0% at two tiers and 92.0% at three; high-aesthetic precision is 90.0%. Two human upper bounds bracket these figures: designer–designer agreement on the high-aesthetic class and annotator–annotator agreement on three-tier usability are both 90.0%, so a model precision at or above the ceiling would indicate a labeling artefact rather than skill. This is the headline: the judge is conservative on the excellent verdict rather than loose with it.

TWO-SIDED USABILITY VALIDATION.

We report the two-tier (excellent / basically usable) and three-tier (excellent / basically usable / unusable) agreement rates together with precision and recall on the excellent tier, so that both directions of error remain visible: a judge that is precise but conservative on the *excellent* verdict is distinguishable from one that is precise and thorough, and the two-tier rate is the headline because it is the distinction the downstream metric actually uses.

ONE-SIDED AESTHETICS VALIDATION.

Three professional designers applied the high-aesthetic standard; their agreement is high on artifacts the standard admits and low within the artifacts it rejects. That asymmetry is a property of the construct rather than a defect of the annotation—the boundary between a page with deliberate visual intent and one without is judgeable, whereas further subdivision inside the rejected set is not. We therefore keep aesthetics as the binary gate of §5 and verify only the positive class, reporting precision as *the fraction of agent-labeled high-aesthetic applications that the designers also rate high-aesthetic*; the negative class is retained as a single bucket rather than validated. The 47-item mean is reported alongside for completeness, but the binary gate, not the mean, is the validated aesthetic signal.

M REWARD AGGREGATION (DETAIL)

The accounting identities are $\#\text{usable} + \#\text{unusable} + \#\text{incomplete} = \#\text{cases}$, and likewise for the aesthetic verdicts. A blank application submits a valid `unavailable_blank` and is counted as negative. The continuous aesthetics summary is the 47-item checklist mean, reported alongside the binary gate for completeness; the gate, not the mean, is the validated aesthetic signal (§5).

N LIMITATIONS

Four limits bound the claims above. *(1) Sampled difficulty.* The suite samples production difficulty rather than replicating its distribution—categories and difficulty tiers are balanced by design, so the reported rates are comparable across strata but their mix is not a model of production traffic. *(2) Versioned aesthetics.* The aesthetic verdict is relative to the rubric version in force; a model's aesthetic rank is a statement about that rubric, and a rubric revision can move ranks even when the underlying artifacts do not change. *(3) Reward-hacking.* The agent, its prompts, and the rubric are released for reuse, which makes gaming the rubric a real risk; we mitigate it with a held-out split and periodic rubric updates rather than treating release as cost-free. *(4) Single backbone.* Both judge policies share one backbone, so what the reliability study validates is this configuration of interactive LLM judging, not LLM judging in general; a different backbone would require re-validating the agreement figures against human labels.