CO-DESIGN BENCH: MULTIMODAL COLLABORATIVE CODING EVALUATION FOR AGENTIC MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating human–AI co-design requires testing whether coding agents can translate real users' multimodal requests into working application revisions. We introduce **Co-Design Bench**, a reference-free benchmark of 273 tasks curated from 37,463 deduplicated queries across 26,314 real user coding sessions. Its two tracks, Multimodal Editing (ME) and Repository Editing (RE), cover localized edits with optional image, audio, video, or document inputs and coordinated repository revisions. Tasks span three capability dimensions: Grounded Understanding, UI Transformation, and Functional Implementation. The tasks pair normalized requests and runnable pre-edit applications with 2,331 hidden expert-authored acceptance criteria. Our *judge-then-refute* protocol grades criteria using source and browser evidence, challenges positive verdicts with a separate LLM verifier, and audits visual and interaction claims without gold code or target screenshots. Under a shared agentic harness, the strongest of ten model endpoints passes an average of 69.9% of criteria per task but fully satisfies only 26.4% of tasks. All ten endpoints achieve lower criterion completion on repository edits spanning three or four change types than on those spanning one or two. Co-Design Bench provides a testbed for studying the gap between partial requirement satisfaction and complete task success under agentic execution for real-user application revisions.

1 INTRODUCTION

Web application co-design follows an iterative cycle of inspecting an existing artifact, requesting changes, and assessing the result (Schön, 1983). Coding agents can participate in this cycle by inspecting visual context, modifying source files, and executing applications (Yang et al., 2024a; Anthropic, 2025; OpenAI, 2026a; Qwen Team, 2026b). Success requires translating user intent into working revisions of the current application. A request may combine image, audio, video, or document inputs with changes to layout, interaction, and application state, requiring the agent to connect multimodal evidence to dependent revisions across the application's interface and behavior.

Existing benchmarks capture important parts of this problem: reference reconstruction (Si et al., 2024; Lai et al., 2025; Chen et al., 2025), synthetic or predefined edits and repairs (Li et al., 2026; Lei et al., 2026), edits extracted from version histories (Xiao et al., 2025), and implementation of repository issues or freelance requests (Jimenez et al., 2023; Yang et al., 2024b; Miserendino et al., 2025). Coverage remains limited for compound revisions drawn from coding sessions across both localized multimodal edits and coordinated repository changes (Table 1). For example, the sports clip editor request in Figure 1(b) couples clip-count and duration limits, timeline alignment under zoom, and state updates that re-enable insertion after clip removal or splitting. Such a request ties success to the behavior of the edited application across several interacting requirements at once.

Agentic execution also changes how benchmark scores should be interpreted. On the same 150 WebCompass-edit tasks (Lei et al., 2026), switching `qwen3.7-plus` (Qwen Team, 2026a) from single-shot to agentic execution raises mean graded feature completion from 73.6% to 95.1%, with the feature rubric, judge, and render pipeline held fixed (Figure 1(a); Appendix A). On these 150 tasks, `opus-4-8` (Anthropic, 2026a) reaches 99.4% mean graded feature completion with agentic execution. The 21.5-point gain shows that execution conditions materially affect measured completion in this setting. We therefore evaluate real-user revisions with iterative tool access and report both criterion satisfaction and complete task success.

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

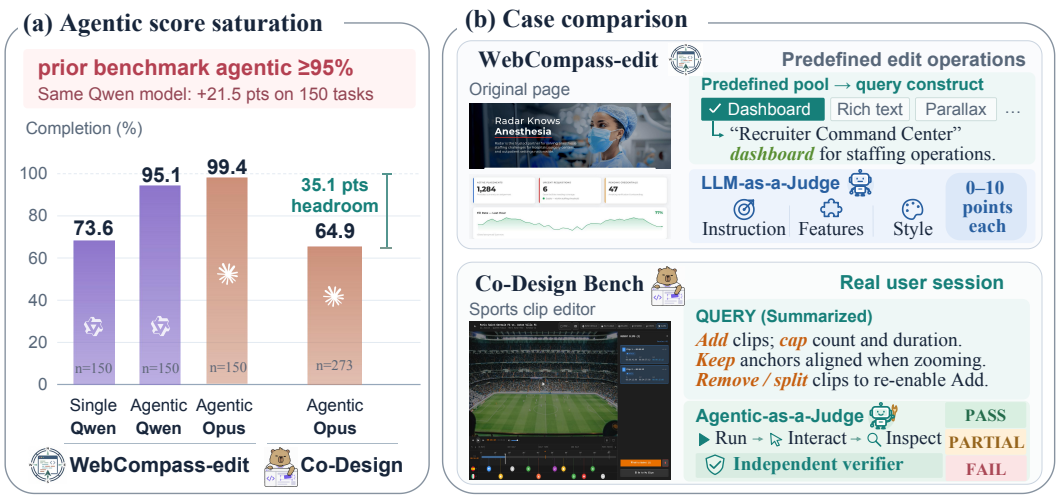

Figure 1: **Execution regimes and task provenance.** **(a)** Mean graded feature completion on 150 WebCompass-edit tasks (Lei et al., 2026): `qwen3.7-plus` rises from 73.6% (single-shot) to 95.1% (agentic), with the feature rubric, judge, and render pipeline held fixed; `opus-4-8` reaches 99.4% with agentic execution (Appendix A). The cross-benchmark bars use different task pools and evaluation protocols. **(b)** A WebCompass request drawn from predefined edit operations and a Co-Design Bench request recovered from a real coding session, with their evaluation workflows.

We introduce **Co-Design Bench**, comprising 273 tasks and 2,331 expert-authored acceptance criteria, curated from 37,463 deduplicated queries across 26,314 real user coding sessions. We recover and validate runnable pre-edit applications, retain available session context and attachments, and consolidate user requests into bounded editing tasks. The benchmark covers two complementary settings: *Multimodal Editing (ME)* evaluates localized visual, structural, and content edits with optional image, audio, video, or document inputs; *Repository Editing (RE)* evaluates coordinated UI and feature changes across components, routes, state, and data flow. Both tracks pair the starting application with a normalized request and hidden acceptance criteria that specify the requested changes while allowing multiple valid implementations. A document-grounded edit can require consistent updates to a displayed value and its dependent summaries, while a repository request can couple visible controls with state transitions across components. The shared task formulation captures these dependencies as auditable requirements, connecting the interpretation of user evidence to observable visual changes and state transitions in the edited application.

Evaluating these tasks requires checking whether requested features are reachable and behave correctly in the edited application. Our *judge-then-refute* protocol first grades hidden criteria using source code and browser execution evidence. A separate LLM verifier challenges PASS and PARTIAL verdicts against the collected evidence, checking for stubs, hardcoding, unreachable functionality, and requirement mismatches. Audits require successful renders and screenshots for visual claims and recorded browser actions for interaction claims; missing evidence aborts evaluation before score aggregation. The pre-edit application grounds the requested changes, enabling reference-free acceptance without gold code or target screenshots, while retained verdicts and traces link each criterion judgment to its supporting evidence for inspection.

We evaluate ten model endpoints under a shared agentic harness. The strongest endpoint passes an average of 69.9% of criteria per task but fully satisfies only 26.4% of tasks. Model rankings vary between tracks, and all ten endpoints achieve lower criterion completion on repository edits spanning three or four change types than on those spanning one or two. These results show that substantial coverage of individual requirements can coexist with limited complete-task success, motivating evaluation of both criterion satisfaction and complete delivery.

Contributions. **(1)** We construct a benchmark of real-user application revisions spanning localized multimodal edits and coordinated repository changes, with runnable starting states and expert-authored criteria. **(2)** We develop a reference-free evaluation protocol that combines code and browser evidence, a separate refutation pass, and mandatory evidence audits to assess requested

Table 1: **Comparison of web-coding benchmarks.** Task origins and core evaluation capabilities of the reported settings.

Benchmark / subset	Task		Evaluation		
	Real-user sessions	Input modalities	Reference-free acceptance	Agentic evaluation	Adversarial verification
REFERENCE RECONSTRUCTION					
Design2Code (Si et al., 2024)	×	I	×	×	×
WebSight-Test (Liang et al., 2024)	×	I	×	×	×
WebRenderBench (Lai et al., 2025)	×	I	×	×	×
IWR-Bench (Chen et al., 2025)	×	I,V	×	×	×
PAGE-LEVEL CODING					
MT-Web2Code (Li et al., 2026)	×	T,I	×	×	×
DesignBench (Xiao et al., 2025)	△	T,I	△	×	×
WebCompass (edit) (Lei et al., 2026)	×	T,I	✓	×	×
REPOSITORY SOFTWARE ENGINEERING					
SWE-bench M (Yang et al., 2024b)	×	T,I,V	△	×	×
SWE-Lancer (IC) (Miserendino et al., 2025)	×	T	✓	×	×
Co-Design Bench (ours)	✓	T, I, V, A, D	✓	✓	✓

✓ =reported; × =not reported; △ =subset/metric-dependent. Sessions denote requests from real-user coding histories, excluding standalone issue reports. DesignBench's session requests belong to its editing subset. T/I/V/A/D =text/image/video/audio/document. Reference-free acceptance needs no gold code or target screenshot; a pre-edit baseline is allowed. Agentic evaluation actively chooses tools/actions, beyond fixed test replay. Adversarial verification separately challenges initial positive verdicts.

changes. **(3)** We characterize the gap between criterion satisfaction and complete task success across ten model endpoints, with analyses of input modality and edit breadth.

2 RELATED WORK

Multimodal web generation and editing. Design2Code, WebSight-Test (from WAFFLE), and WebRenderBench evaluate screenshot-to-code reconstruction against reference pages (Si et al., 2024; Liang et al., 2024; Lai et al., 2025). IWR-Bench extends reconstruction to interaction videos, using annotated action replay and reference screenshots (Chen et al., 2025). WebGen-Bench and FrontendBench add browser- or sandbox-based functional tests for websites generated from scratch (Lu et al., 2025; Zhu et al., 2025). Editing benchmarks instead revise an existing artifact. DesignBench extracts adjacent-version pairs from V0/Vue0 histories for before/after screenshot assessment (Xiao et al., 2025). MT-Web2Code synthesizes repair trajectories by corrupting reference pages (Li et al., 2026), while WebCompass constructs tasks around predefined edit operations (Lei et al., 2026). FronTalk studies conversational front-end generation with multimodal feedback (Wu et al., 2026). Co-Design Bench instead fixes a bounded revision recovered from a production coding session, retaining its runnable starting application, context, and attachments; hidden criteria specify requested changes without gold code or a target screenshot as supervision.

Repository-level coding. SWE-bench evaluates changes for real GitHub issues through executable tests (Jimenez et al., 2023). SWE-bench Multimodal covers frontend JavaScript software with visual tests (Yang et al., 2024b), and SWE-Lancer evaluates real freelance implementation requests through browser-based end-to-end tests (Miserendino et al., 2025). Co-Design Bench focuses on compound design and feature revisions within coding sessions. Its repository track covers coordinated changes to components, routes, state, and data flow. A shared task interface and acceptance protocol connect this setting to localized multimodal edits, allowing both requirement coverage and complete delivery to be measured on a common scale across the two tracks.

Agentic artifact evaluation and verification. Agent-as-a-Judge assesses generated artifacts against user requirements using agent capabilities (Zhuge et al., 2024). WebCompass uses browser interaction and adaptive tests for generation; its editing and repair settings use checklist-guided judgments from code, logs, and screenshots (Lei et al., 2026). Structured LLM evaluation can improve agreement with human judgments, yet evaluators can exhibit self-preference and benchmark-dependent reliability (Liu et al., 2023; Panickssery et al., 2024; Lù et al., 2025). For application revisions, source presence alone does not establish reachability or correct behavior. Our *judge-then-*

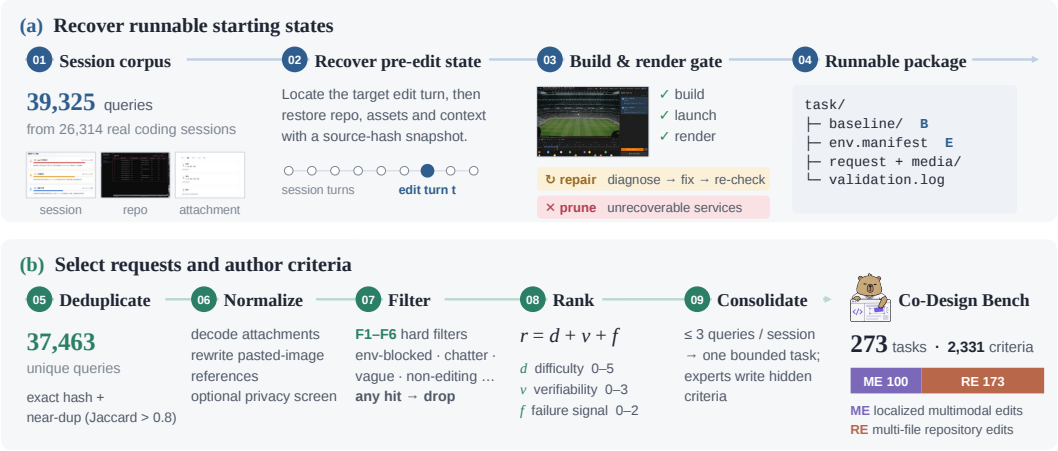

Figure 2: **Data construction and filtering.** (a) Recover and validate runnable pre-edit applications, repairing eligible environments and excluding unrecoverable cases. (b) Deduplicate and filter requests, rank candidates by difficulty, verifiability, and historical failure signals, then select bounded session intervals for consolidation and expert rubric authorship.

refute protocol therefore combines source and browser evidence with a separate verifier; mandatory audits require screenshots for visual claims and recorded actions for interaction claims. Retained evidence makes positive credit traceable, though it cannot eliminate judgment error.

3 Co-Design Bench

Co-Design Bench evaluates the transition from a runnable application to an edited application in response to a real user's revision request. Its 273 tasks span Multimodal Editing (ME) and Repository Editing (RE), with 2,331 expert-authored acceptance criteria. The tracks share an evaluation contract while covering two editing scales: localized changes grounded in multimodal evidence and coordinated repository changes spanning structure, behavior, and content.

3.1 Task Formulation

Each task is $t = (S, I, C)$, where $S = (B, E)$ contains a runnable pre-edit baseline B and its containerized environment E. The input $I = (q, X, K)$ comprises a normalized request q, optional multimodal attachments X, and retained session history K. The hidden expert-authored rubric $C = C_t = \{c_i\}_{i=1}^{n_t}$ specifies the requested outcomes and any explicit architectural constraints while permitting any implementation consistent with those task-specific requirements.

Under a solver harness H, an agent $\mathcal{A}_\theta^H$ edits the application through an execution trajectory τ:

$$\mathcal{A}_\theta^H : (S, I) \longmapsto (\widehat{S}, \tau), \qquad \widehat{S} = \mathrm{Edit}_\tau(S). \tag{1}$$

The solver can inspect, edit, build, and execute the application through H; criteria and evaluation artifacts remain hidden. Acceptance concerns the final application state. The baseline grounds what must change and any requested preservation constraints, so reference-free acceptance remains compatible with inspecting the pre-edit artifact. Section 4 defines the completion metrics.

3.2 Data Source and Curation

The source corpus contains 39,325 production queries across 26,314 coding sessions; deduplication retains 37,463 candidates. Each candidate is anchored to a target edit turn with its repository, available context, and attachments. Figure 2 summarizes the construction pipeline.

Recovering executable starting states. We reconstruct applications from session logs, repository archives, or HTML bundles. Candidates must pass dependency, build, container launch, and Playwright rendering checks (Microsoft, 2026). Tasks requiring unrecoverable services are excluded,

and repaired applications pass the same validation gate. This gate establishes a working starting point, allowing evaluation to assess the changes made during solver execution.

Selecting requests and authoring acceptance criteria. Normalization packages attachments and clarifies request wording. Hard filters remove environment-blocked, non-actionable, and non-editing requests. Remaining candidates are ranked by structural difficulty, verifiability, and historical failure signals such as reformulations and rollbacks, then consolidated into bounded session intervals. Experts refine the requests and author hidden criterion-level rubrics. The resulting tasks preserve session-derived revision goals but evaluate a fixed request; they do not simulate further user feedback during execution. Appendix I gives filtering thresholds, ranking rules, and selection parameters.

3.3 Multimodal Editing (ME)

The 100 ME tasks cover localized visual, structural, and content edits. The solver receives a runnable page, an annotated target region, a normalized request, and optional image, audio, video, or document attachments, available for direct inspection or tool-based processing. These assets can specify a visual treatment, a spoken instruction, or a value to reconcile across views. Validation checks the initial Playwright render, attachment decodability, target-region annotations, and criterion verifiability. Rubrics specify requested visual treatments, content placement, and spatial relationships, including preservation of unrelated content when requested. Different CSS and DOM implementations can satisfy these outcomes. For example, the receipt-grounded case requires reconciling a value across a source document, an editable row, and dependent summaries: the local edit must preserve consistency across the requested views (Appendix E).

The modality split comprises 20 tasks without attachments, 4 with images, 27 with video, 32 with audio, and 17 with documents. These tasks contain 553 criteria, with a mean of 5.5, a median of 6, and a maximum of 11 per task. See Table 8 for more detail.

3.4 Repository Editing (RE)

The 173 RE tasks start from runnable multi-file frontend repositories and require coordinated changes across components, routes, state, data flow, or content. Each task pairs a consolidated request with repository context and relevant local assets. Experts decompose compound requests into separately auditable criteria, retaining explicit wiring or artifact constraints while leaving other implementation choices open. Validation checks dependency resolution, builds, runtime launch, route reachability, and browser rendering; tasks requiring unrecoverable services are excluded. In the clip-editor case, controls must respect duration and count limits and recover after removing or splitting clips, coupling visible interface changes to application state (Appendix E). Both tracks use the same (S, I, C) interface and PASS/PARTIAL/FAIL verdicts. They differ in the requested editing scope: a localized edit may also start from a bundled application. Appendix F.1 details dataset composition, and Table 1 compares existing web-coding settings.

This track contains 1,778 criteria, averaging 10.3 per task, with a median of 8 and a maximum of 46. Structural, stylistic, interactive, and content edits occur in 151, 133, 131, and 112 tasks, respectively. These counts overlap: 166 of the 173 tasks require multiple edit types, making coordinated changes a common property of the selected requests (Table 8).

4 Evaluation Protocol

Evaluation assesses the final application W against the request and hidden criteria, using baseline B to ground the changes. Figure 3 shows the *judge-then-refute* pipeline.

4.1 Evidence and Adversarial Verification

Evidence and primary grading. The candidate is built and rendered in an isolated environment. A fixed agentic evaluator (Zhuge et al., 2024) assigns PASS, PARTIAL, or FAIL to each criterion using source diffs, rendered pages, and Playwright traces. Evidence must match the claim: source inspection can establish a static property, a screenshot can support a visual requirement, and recorded actions can establish an interaction and its resulting state. Failed builds or missing DOM nodes

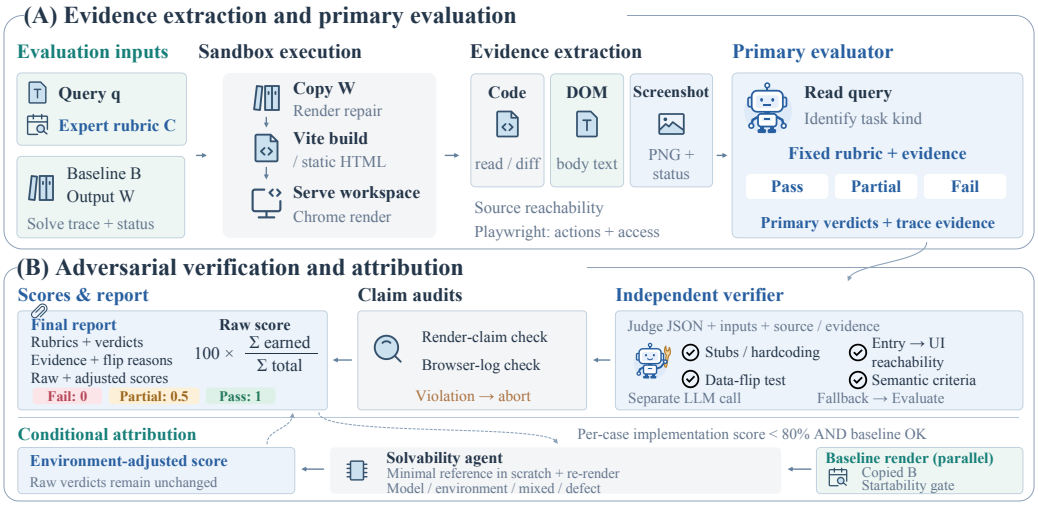

Figure 3: **Evaluation from artifact to verdict.** (a) Build and render the candidate, collect source and browser evidence, and grade hidden criteria. (b) Challenge positive verdicts, audit required evidence, and aggregate scores. Parallel baseline and solvability checks diagnose failures for supplementary attribution analysis.

invalidate render-dependent criteria; a handler in source code alone cannot establish that users can reach and operate the requested functionality through the rendered interface.

Refutation and evidence audits. A separate LLM verifier receives the initial judgments and collected evidence, then challenges PASS and PARTIAL verdicts. It checks for stubs, hardcoded outputs, unreachable entry points, failures under varied inputs, and request mismatches. This pass targets unsupported completion credit; it does not by itself establish the correctness of every verdict. Subsequent audits require successful renders and screenshots for visual claims and browser-action logs for interaction claims. Violations abort evaluation before aggregation. Verdicts, evidence, and refutation logs are retained for inspection, linking each reported outcome to its supporting observations. Two frontend experts additionally checked rubric scores for 200 task–model outputs. They examined the task requirements, evaluation rubric, and generated interface, including browser interactions when required, to assess whether each PASS/PARTIAL/FAIL verdict reflected the output's satisfaction of the corresponding criterion. Appendix J specifies execution, evidence extraction, and artifact retention.

4.2 SCORING AND FAILURE ATTRIBUTION

For task t with n_t implementation criteria, let v_{ti} denote the verdict after refutation and auditing. Strict and Graded Completion are

$$R_t = \frac{1}{n_t}\sum_{i=1}^{n_t} \mathbf{1}[v_{ti} = \text{PASS}], \qquad G_t = \frac{1}{n_t}\sum_{i=1}^{n_t} g(v_{ti}), \tag{2}$$

where $\mathbf{1}[\cdot]$ is the indicator function and $g(\text{PASS}) = 1$, $g(\text{PARTIAL}) = 0.5$, and $g(\text{FAIL}) = 0$. Over a fixed task set $\mathcal{T}$ shared by all endpoints, we report equal-weight task means and Full-Pass Rate:

$$\bar{R} = \frac{\sum_{t\in\mathcal{T}} R_t}{|\mathcal{T}|}, \quad \bar{G} = \frac{\sum_{t\in\mathcal{T}} G_t}{|\mathcal{T}|}, \quad \text{F} = \frac{\sum_{t\in\mathcal{T}} \mathbf{1}[R_t = 1]}{|\mathcal{T}|}. \tag{3}$$

Task normalization prevents longer rubrics from dominating the mean. Strict Completion measures fully satisfied requirements, Graded Completion credits partial progress, and Full-Pass Rate measures delivery of every criterion in a task. On the normalized $[0, 1]$ scale, $\text{F} \le \bar{R} \le \bar{G}$; the gap $\bar{R} - \text{F}$ is criterion credit from tasks with unfinished requirements. A high $\bar{R}$ can therefore coexist with a low F when unresolved requirements are spread across many tasks. Each endpoint contributes 100 ME and 173 RE tasks; combined scores weight all 273 tasks equally.

Failure attribution. For low-scoring cases ($G_t < 0.8$), attribution proceeds when an independently rendered baseline passes the startability gate. A parallel solvability probe attempts a minimal patch

in an isolated sandbox, stubbing inaccessible network endpoints while preserving core business logic. The checks distinguish model limitations, environmental blockage, mixed causes, and defects in the starting artifact. Raw scores and denominators remain unchanged. Supplementary adjusted scores grant attribution credit only when infrastructure failures are established (Appendix J.1).

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Shared solver harness. The main comparison holds the Claude Code harness (Anthropic, 2025), container manifests, system prompts, tool interfaces, and repository staging procedures fixed across endpoints. Each solver receives the normalized request, attachments, and a clean working copy; acceptance criteria remain hidden. After termination, the final application is built, rendered, and evaluated under Section 4. Runs are capped at three hours and 120 turns. Reaching the turn cap triggers evaluation of the final state; wall-clock timeouts are recorded as infrastructure failures, while raw scores and denominators remain unchanged (Appendices D and J.1).

Models, coverage, and metrics. We evaluate ten model endpoints on the fixed 273 tasks. The complete grid contains 2,730 model–task pairs: 1,000 in ME and 1,730 in RE. Table 2 reports per-track results, and Appendix B gives model references and the combined leaderboard. We report Strict Completion ($\bar{R}$), Graded Completion ($\bar{G}$), and Full-Pass Rate (F), as defined in Section 4.2. Completion scores use the $[0, 1]$ scale unless noted, while full-pass rates are percentages. Task-group analyses pool final-state scores within each group across all ten model endpoints.

5.2 MAIN RESULTS

Table 2 reports completion, execution effort, and tool use across endpoints. The strongest model, `gpt-6-astra`, leads both tracks in mean strict completion. The remaining endpoints vary in their relative strengths, while complete task success remains limited even for the highest-scoring model.

Criterion coverage exceeds complete delivery. The leading endpoint achieves $\bar{R} = 0.754$ on ME and 0.667 on RE, but fully passes only 41.0% and 17.9% of tasks, respectively. Across all 273 tasks, its mean strict completion is 0.699 and full-pass rate is 26.4%. Thus, 73.6% of tasks retain at least one criterion that is not fully satisfied. Reporting both metrics exposes incomplete delivery because average requirement coverage does not directly measure full-task success.

Rankings differ across editing settings. Strict-completion rankings correlate across tracks (Spearman $\rho = 0.818$), yet `kimi-k3` moves from fourth on ME to sixth on RE, while `Qwen3.8-27B` moves from fifth to second. The latter achieves $\bar{R} = 0.645$ on repository edits, compared with 0.642 for `opus-5`, making it the second-ranked endpoint in this setting. It ranks fourth overall at 0.644, following `gpt-6-astra` (0.699), `opus-5` (0.661), and `qwen3.8-max-0902` (0.653; Table 5). The small repository-track gap does not establish a general model advantage. Endpoint mean strict scores span 28.7 percentage points on ME and 13.0 on RE; a combined ranking can conceal strengths specific to an editing setting within this benchmark.

Execution profiles differ across tracks. ME video tasks have a median of 44 turns, compared with 27 for text-only tasks. Media interactions account for 82.8% of activity in the upper-quartile longest rollouts. Image-inspection calls average 14.47 per run on ME and 2.78 on RE, while the share of `Edit`/`Write` calls rises from 7.3% to 27.6%. These profiles describe how agents allocate execution effort under the shared harness; they do not isolate the causal effects of perception or editing tools on task completion because task content differs across these comparisons (Appendix G).

Execution effort and completion. On ME, `gpt-6-astra` reaches $\bar{R} = 0.754$ with 32 median turns and 1.24M tokens per case, whereas `GLM-5.2` uses 69 turns and 7.29M tokens for $\bar{R} = 0.572$. On RE, `Qwen3.8-27B` uses 57 turns and 6.72M tokens for $\bar{R} = 0.645$, compared with 48 turns and 2.33M tokens for `gpt-6-astra` at $\bar{R} = 0.667$. Higher completion therefore does not require uniformly greater execution effort across the tested endpoints. Token traffic follows the API accounting conventions in Appendix D, including retries and reported cache usage; it is an execution measure rather than a common monetary cost.

Table 2: **Main benchmark performance across ME and RE.** Ranked by mean strict completion $\bar{R}$. Metrics report Strict ($\bar{R}$), Graded ($\bar{G}$), and Full-Pass Rate (F, in %). Rollout statistics summarize median turns, mean turn output tokens, and total token traffic per case. Tool distributions are grouped into execution (`Bash`), inspection (`Read/Grep/Glob`), modification (`Edit/Write`), and auxiliary. Bold indicates peak track performance.

Model	Completion			Rollout statistics			Tool calls (%)			
	Strict ↑	Graded ↑	F ↑	Turns	Len.	Tokens	Exec	Insp.	Mod.	Overall
ME n=100										
gpt-6-astra	**0.754**	**0.798**	**41.0**	32	146	1.24M	36.8	34.7	13.7	
opus-5	0.695	0.749	30.0	34	363	2.92M	55.8	35.8	7.8	
qwen3.8-max-0902	0.667	0.719	31.0	42	620	2.94M	46.9	33.0	7.8	
kimi-k3	0.662	0.718	26.0	27	329	1.47M	47.9	40.0	7.9	
qwen3.8-27b	0.642	0.705	23.0	56	845	5.90M	45.8	35.7	6.6	
opus-4-8	0.593	0.657	16.0	24	560	1.65M	49.8	37.8	9.9	
gpt-5.6-sol	0.583	0.639	20.0	32	208	1.54M	34.1	35.7	9.5	
glm-5.2	0.572	0.629	16.0	69	659	7.29M	74.6	13.1	4.9	
glm-5.3-flash	0.538	0.599	18.0	49	357	3.65M	53.6	40.7	3.7	
grok-4.6	0.467	0.536	6.0	14	491	0.89M	26.8	62.6	10.7	
RE n=173										
gpt-6-astra	**0.667**	**0.728**	**17.9**	48	272	2.33M	20.3	31.1	37.6	
qwen3.8-27b	0.645	0.711	10.4	57	1185	6.72M	32.8	28.0	22.3	
qwen3.8-max-0902	0.644	0.715	8.1	45	893	4.01M	32.7	22.5	28.3	
opus-5	0.642	0.702	11.6	35	697	3.31M	44.4	19.2	34.5	
gpt-5.6-sol	0.627	0.702	11.0	40	262	1.80M	20.8	36.8	19.6	
kimi-k3	0.610	0.681	9.8	24	534	1.36M	34.3	27.4	30.7	
glm-5.3-flash	0.594	0.662	6.4	31	481	1.86M	43.0	21.6	29.6	
opus-4-8	0.575	0.645	6.9	23	815	1.77M	27.6	25.3	32.5	
glm-5.2	0.564	0.641	5.8	53	603	3.28M	38.7	24.2	18.2	
grok-4.6	0.536	0.605	6.9	9	1157	0.45M	1.8	61.9	36.2	

5.3 Task-Level Analysis

We examine three complementary task groupings: attachment modality on ME, edit breadth on RE, and primary capability across both tracks. These descriptive comparisons reflect the selected requirements and composition of each task group rather than isolated causal effects.

Attachment modality. `Qwen3.8-27B` leads text-only tasks ($\bar{R} = 0.779$), `opus-5` leads image tasks (0.891), and `gpt-6-astra` leads video, audio, and document tasks (Figure 4a). Pooled mean strict completion ranges from 0.584 to 0.652 across modalities. Documents have the highest pooled strict score (0.652), yet only 18.2% of their model–task pairs fully pass, compared with 37.0% for text-only tasks (Table 6a). Criterion coverage and all-pass success thus give different views of task difficulty. Tool traces document diverse processing strategies: OpenCV (Bradski, 2000) appears in 68.5% of video trajectories, `ffmpeg` (FFmpeg Developers, 2026) in 31.5% of video and 25.0% of audio runs, and `faster-whisper` (Klein, 2023) in 4.4% of audio runs. Some agents also execute pretrained perception models to obtain descriptions, transcripts, class scores, or detection boxes from media inputs to ground subsequent edits (Appendix G).

Edit breadth. On RE, overlapping groups of structural, stylistic, interactive, and content edits have similar pooled strict scores (0.599–0.607). Grouping by the number of edit types reveals a larger difference: tasks with one or two types score 0.654, versus 0.591 for three or four types, a 6.3-point gap. Every endpoint scores lower on the broader group (Figure 4b). Full-pass rates remain similar (9.1% versus 9.7%), while zero-pass rates rise from 0.9% to 5.8% (Table 6b). Broader edits have lower criterion coverage while maintaining similar pooled full-pass rates.

Primary capability. Table 3 groups all 2,730 evaluation pairs by the task's main delivery objective. Functional Implementation is the largest group (126 tasks, 46.2%). It has the highest pooled strict completion (0.619) but the lowest full-pass rate (10.3%), compared with Grounded Understanding (0.604; 20.2%) and UI Transformation (0.615; 13.6%). The distinction between requirement coverage and complete delivery persists across these groups. `gpt-6-astra` leads all three di-

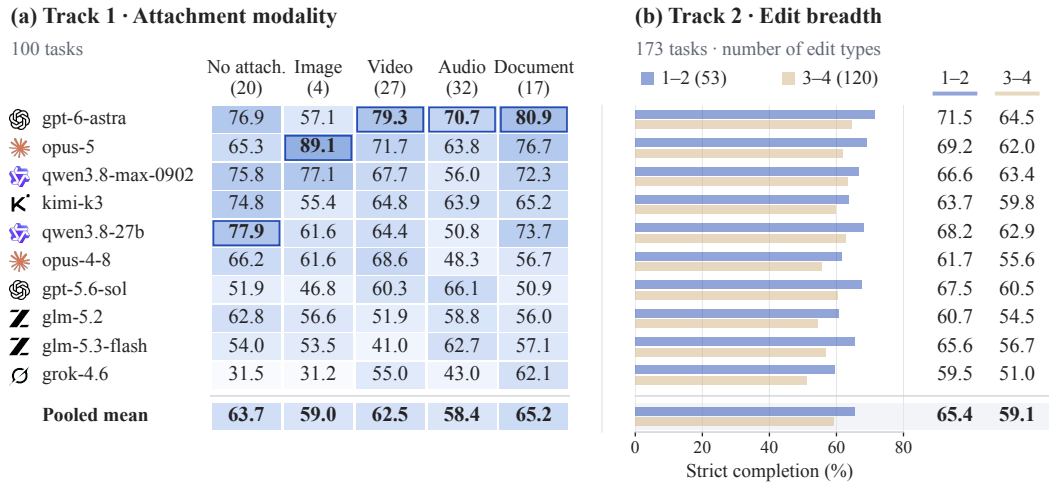

Figure 4: **Mean strict completion ($\bar{R}$, in %) across task strata ($N = 273$).** Ten endpoints are ordered by ME mean strict completion. (a) Five input attachment modalities; bold outlines mark column maxima. (b) Tasks with 1–2 edit types (blue) versus 3–4 (gold). Parentheses give task counts. Pooled rows average all task scores across ten endpoints (Appendix B.2).

		Ten endpoints		
Primary capability	**Tasks**	**Pairs**	**Strict**	**Full (%)**
Grounded Understanding	92	920	0.604	20.2
UI Transformation	55	550	0.615	13.6
Functional Implementation	126	1260	0.619	10.3

Table 3: **Primary capability breakdown.** Ten endpoints on 273 tasks give 2,730 evaluation pairs. Strict averages task pass fractions; Full is the all-pass rate. Figure 9 gives scores for each model and capability group.

mensions, scoring 0.738, 0.692, and 0.673 on Grounded Understanding, UI Transformation, and Functional Implementation, respectively; `Qwen3.8-27B` reaches 0.661 on the latter. These groups overlap strongly with track membership, so the breakdown describes performance rather than isolating independent abilities. The accompanying multi-label annotation records 1,172 task–capability associations: 265 tasks (97.1%) have multiple capability labels, and 174 (63.7%) span more than one high-level dimension. Appendix F details category results and coverage.

5.4 Ablation and Robustness Studies

We conduct systematic ablations across three critical experimental axes. **Execution Regimes:** Comparing single-shot against agentic execution on 150 WebCompass-edit tasks demonstrates the necessity of iterative agentic loops (Appendix A). **Evaluator Sensitivity:** Swapping the underlying judge model confirms that relative model rankings and score distributions remain highly robust across different evaluators (Appendix C). **Scaffold Robustness:** Evaluating identical solvers across three distinct harnesses reveals strong model×harness interactions and highlights the decisive role of closed-loop verification (Appendix H).

6 Conclusion

Co-Design Bench evaluates 273 real-user application revisions with runnable starting states and 2,331 hidden acceptance criteria. Its two tracks span localized multimodal edits and coordinated repository changes under a shared protocol of source evidence, browser execution, and adversarial review. Across ten endpoints, the strongest model reaches 69.9% mean strict completion but fully satisfies only 26.4% of tasks. Track-specific rankings, edit-breadth differences, and harness sensitivity show where reliable delivery remains difficult. Runnable baselines and criterion-level evidence support analysis of failures in request grounding, change propagation, and final-state verification. The benchmark provides a common setting for measuring progress toward complete delivery by coding agents, while retaining the task context and execution evidence needed to analyze unmet requirements and compare future solvers.

REFERENCES

Anomaly. OpenCode. https://github.com/anomalyco/opencode/releases/tag/v1.18.13, 2026. Software version 1.18.13; accessed 2026-09-23.

Anthropic. Claude Code. https://code.claude.com, 2025. Agentic coding tool. Accessed 2026-08-28.

Anthropic. Introducing Claude Opus 4.8. https://www.anthropic.com/news/claude-opus-4-8, May 2026a. Published May 28, 2026; accessed 2026-09-23.

Anthropic. Introducing Claude Opus 5. https://www.anthropic.com/news/claude-opus-5, July 2026b. Published July 24, 2026; accessed 2026-09-23.

Gary Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 2000.

Yang Chen, Minghao Liu, Yufan Shen, Yunwen Li, Tianyuan Huang, Xinyu Fang, Tianyu Zheng, Wenxuan Huang, Cheng Yang, Daocheng Fu, Jianbiao Mei, Rong Wu, Yunfei Zhao, Licheng Wen, Xuemeng Yang, Song Mao, Qunshu Lin, Zhi Yu, Yongliang Shen, Yu Qiao, and Botian Shi. IWR-Bench: Can LVLMs reconstruct interactive webpage from a user interaction video?, 2025. arXiv:2509.24709.

FFmpeg Developers. FFmpeg. https://ffmpeg.org/, 2026. Software; accessed 2026-09-23.

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?, 2023. arXiv:2310.06770; ICLR 2024.

Kimi Team. Kimi K3: Open frontier intelligence. https://arxiv.org/abs/2607.24653, 2026. arXiv:2607.24653.

Guillaume Klein. faster-whisper: Faster whisper transcription with CTranslate2. https://github.com/SYSTRAN/faster-whisper, 2023. Software; accessed 2026-09-23.

Peichao Lai, Jinhui Zhuang, Kexuan Zhang, Ningchang Xiong, Shengjie Wang, Yanwei Xu, Chong Chen, Yilei Wang, and Bin Cui. WebRenderBench: Enhancing web interface generation through layout-style consistency and reinforcement learning, 2025. arXiv:2510.04097.

Xinping Lei, Xinyu Che, Junqi Xiong, Chenchen Zhang, Yukai Huang, Chenyu Zhou, Haoyang Huang, Minghao Liu, Letian Zhu, Hongyi Ye, Jinhua Hao, Ken Deng, Zizheng Zhan, Han Li, Dailin Li, Yifan Yao, Ming Sun, Zhaoxiang Zhang, and Jiaheng Liu. WebCompass: Towards multimodal web coding evaluation for code language models, 2026. arXiv:2604.18224.

Qiming Li, Shujie Hu, Haohan Liu, Xiaocheng Feng, Songxiang Liu, and Guanglu Wan. MT-Web2Code: Benchmarking coding agents on multi-turn regional reconstruction and localized modification, 2026. arXiv:2608.03474.

Shanchao Liang, Nan Jiang, Shangshu Qian, and Lin Tan. WAFFLE: Finetuning multi-modal models for automated front-end development, 2024. arXiv:2410.18362.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.153. URL https://aclanthology.org/2023.emnlp-main.153/.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories, 2025. arXiv:2504.08942.

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, 2025. arXiv:2505.03733.

Microsoft. Playwright. https://playwright.dev/, 2026. Documentation; accessed 2026-09-23.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. SWE-Lancer: Can frontier LLMs earn $1 million from real-world freelance software engineering?, 2025. arXiv:2502.12115.

OpenAI. Codex CLI. https://github.com/openai/codex/releases/tag/rust-v0.142.0, 2026a. Software version 0.142.0; accessed 2026-09-23.

OpenAI. Previewing GPT-5.6 Sol: A next-generation model. https://openai.com/index/previewing-gpt-5-6-sol/, June 2026b. Official model announcement.

OpenAI. GPT-6 Astra: A new generation of intelligence. https://openai.com/index/gpt-6-astra/, September 2026c. Official model announcement.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations, 2024. arXiv:2404.13076.

Qwen. Qwen Code. https://github.com/QwenLM/qwen-code/releases/tag/v0.21.11, 2026. Software version 0.21.11; accessed 2026-09-23.

Qwen Team. Qwen3.7-Plus: Multimodal agent intelligence. https://qwen.ai/blog?id=qwen3.7-plus, 2026a. Official model release blog.

Qwen Team. Qwen3.8-27B. https://modelscope.cn/models/Qwen/Qwen3.8-27B, 2026b. Official model card; accessed 2026-09-23.

QwenCloud. Qwen3.8-Max-0902. https://www.qwencloud.com/models/qwen3.8-max-0902, n.d. Official model card; accessed 2026-09-23.

Donald A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, 1983.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2Code: Benchmarking multimodal code generation for automated front-end engineering, 2024. arXiv:2403.03163; NAACL 2025.

Xueqing Wu, Zihan Xue, Da Yin, Shuyan Zhou, Kai-Wei Chang, Nanyun Peng, and Yeming Wen. FronTalk: Benchmarking front-end development as conversational code generation with multimodal feedback, 2026. arXiv:2601.04203; CoLM 2026.

xAI. Grok 4.6. https://x.ai/news/grok-4-6, August 2026. Published August 12, 2026.

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, 2025. arXiv:2506.06251.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024a. doi: 10.52202/079017-1601.

John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. SWE-bench multimodal: Do AI systems generalize to visual software domains?, 2024b. arXiv:2410.03859; ICLR 2025.

Z.ai. GLM-5.2: Built for long-horizon tasks. https://z.ai/blog/glm-5.2, June 2026a. Official model release blog.

Z.ai. GLM-5.3-Flash: Frontier intelligence, flash cost. https://z.ai/blog/glm-5.3-flash, August 2026b. Official model release blog.

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, 2025. arXiv:2506.13832.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-Judge: Evaluate agents with agents, 2024. arXiv:2410.10934.

APPENDIX

AI USE STATEMENT

During manuscript preparation, we used LLMs only for language polishing. The authors take responsibility for the final content of this paper.

OUTLINE

Unless stated otherwise, the analyses below use the task sets and evaluation grid in Section 5.1. Appendix A reports the paired WebCompass-edit comparison. Appendices B and C give additional completion results and evaluator sensitivity checks. Appendix D specifies execution limits and resource accounting, and Appendix E illustrates tasks from both tracks. Appendix F details dataset composition and capability coverage. Appendices G and H examine tool use and harness sensitivity. Appendices I and J document curation and evaluation procedures.

A EFFECT OF SOLVER EXECUTION ON WEBCOMPASS-EDIT

Paired design and execution regimes. We use WebCompass-edit to examine how the solver's execution regime affects measured completion. The comparison evaluates `qwen3.7-plus` on the same 150 tasks from the `editing/sp` split (Lei et al., 2026), holding the feature rubric, Opus-class vision judge, and rendering pipeline fixed. The single-shot solver receives the source code, rendered page, and edit request, and returns search/replace blocks in one response. The agentic solver can inspect the application, modify files, and execute code iteratively. We convert its final files to the same search/replace format and verify that reapplying the edits reproduces the final artifact for rendering and grading.

Feature-level scoring. Each requested feature receives a PASS, PARTIAL, or FAIL verdict, scored as 1, 0.5, or 0. We average feature scores within each task and then average the 150 task scores with equal weight. We also report the number and fraction of all-pass tasks, for which every feature receives PASS. This re-evaluation uses graded feature completion in place of WebCompass-edit's original 0–10 dimensional ratings (Lei et al., 2026).

Table 4: **Paired execution-regime comparison on 150 WebCompass-edit tasks.** Both regimes use `qwen3.7-plus` and the same feature rubric, vision judge, and rendering pipeline. Mean completion averages task-level graded scores; an all-pass task satisfies every feature.

Regime	Mean graded completion (%)	All-pass tasks	Full-pass rate (%)
Single-shot	73.6	49	32.7
Agentic	95.1	104	69.3

Results and scope. Agentic execution raises mean graded completion from 73.6% to 95.1%, a gain of 21.5 percentage points (Table 4). All-pass tasks increase from 49 to 104, corresponding to full-pass rates of 32.7% and 69.3%. The remaining 46 tasks retain at least one feature below PASS, despite the high mean graded score. The paired result describes the change between the tested execution regimes for this endpoint and task split. It motivates evaluating revision requests with iterative tool access and reporting both criterion completion and complete task success. Co-Design Bench applies this evaluation focus to multimodal requests recovered from real coding sessions. Figure 1(a) additionally reports 99.4% mean graded feature completion for `opus-4-8` under agentic execution on the same 150 WebCompass-edit tasks. The cross-benchmark scores in Figure 1 involve different task pools and evaluation protocols; the paired WebCompass comparison concerns execution within a fixed task set.

B Additional Results

B.1 Combined Completion

Table 5 reports Strict Completion, Graded Completion, and Full-Pass Rate across all 273 tasks for each endpoint. Each task receives equal weight, so the combined result reflects the 100 ME and 173 RE tasks in proportion to their counts. Table 2 reports the two tracks separately.

Table 5: **Combined performance across both tracks.** Strict ($\bar{R}$) and Graded ($\bar{G}$) weight each task equally over the full benchmark; Full (F) is the all-pass task rate. Rows are ranked by $\bar{R}$. Per-track results are in Table 2.

Model	Strict ↑	Graded ↑	Full (%) ↑
`gpt-6-astra` (OpenAI, 2026c)	**0.699**	0.753	26.4
`opus-5` (Anthropic, 2026b)	0.661	0.719	18.3
`qwen3.8-max-0902` (QwenCloud, n.d.)	0.653	0.716	16.5
`qwen3.8-27b` (Qwen Team, 2026b)	0.644	0.709	15.0
`kimi-k3` (Kimi Team, 2026)	0.629	0.695	15.8
`gpt-5.6-sol` (OpenAI, 2026b)	0.611	0.679	14.3
`opus-4-8` (Anthropic, 2026a)	0.581	0.649	10.3
`glm-5.3-flash` (Z.ai, 2026b)	0.573	0.639	10.6
`glm-5.2` (Z.ai, 2026a)	0.567	0.637	9.5
`grok-4.6` (xAI, 2026)	0.511	0.580	6.6

`gpt-6-astra` leads all three combined metrics. `Qwen3.8-27B` ranks fourth overall and second on RE. Reading the combined and per-track results together distinguishes overall completion from strengths specific to an editing setting.

B.2 Modality and Edit Scope

Table 6 pools results across the ten endpoints by ME attachment modality and RE edit breadth. The final two rows of panel (b) correspond to Figure 4b. Table 9 cross-tabulates attachment modality and starting-artifact structure to clarify how these task attributes overlap.

Table 6: **Completion by input modality and edit breadth across ten endpoints.** Strict ($\bar{R}$) and Graded ($\bar{G}$) average task scores; verdict shares pool criterion judgments. Full and Zero are task rates (%) with $R_t = 1$ and $R_t = 0$.

(a) ME

Modality	Completion		Verdicts (%)			Outcomes	
	Strict	Graded	Pass	Partial	Fail	Full	Zero
None	0.637	0.672	64.8	6.7	28.4	37.0	14.5
Image	0.590	0.731	61.5	25.4	13.1	10.0	2.5
Video	0.625	0.691	62.9	13.1	24.0	19.6	1.5
Audio	0.584	0.652	58.9	13.2	27.9	20.3	0.3
Document	0.652	0.682	66.5	6.2	27.3	18.2	7.1

(b) RE

Edit types	Strict	Full	Zero
1	0.612	11.4	0.0
2	0.661	8.7	1.1
3	0.594	8.7	2.9
4	0.589	10.4	8.1
1–2	0.654	9.1	0.9
3–4	0.591	9.7	5.8

Document tasks have the highest pooled Graded Completion, while tasks without attachments have the highest Full-Pass Rate. In RE, tasks spanning three or four edit types have lower Strict Completion and more zero-pass outcomes than those spanning one or two types, with similar full-pass rates. These groups differ in their requests and contexts as well as their labels; the comparisons describe the benchmark rather than estimate controlled effects of modality or edit breadth.

C Evaluator Sensitivity Check

Setup. We rescore stored outputs from `kimi-k3` and `Qwen3.8-27B` using three alternative judges: `GLM-5.2`, `kimi-k3`, and `Qwen3.8-27B`. Both solvers are evaluated on the full bench-

mark under each configuration. They are selected because their ordering reverses between tracks in the main results, providing a concrete comparison whose sensitivity can be checked.

Table 7: **Graded completion under four evaluator configurations.** All evaluators score the same released tasks in each track. Parentheses give changes from production.

Endpoint	Track	Production	glm-5.2	kimi-k3	Qwen3.8-27B
`kimi-k3`	1	0.718	0.699 (−0.019)	0.737 (+0.018)	0.732 (+0.013)
	2	0.681	0.641 (−0.040)	0.676 (−0.005)	0.662 (−0.019)
`qwen3.8-27b`	1	0.705	0.666 (−0.039)	0.706 (+0.001)	0.722 (+0.017)
	2	0.711	0.658 (−0.053)	0.702 (−0.009)	0.686 (−0.025)

Results. As summarized in Table 7, absolute graded scores shift by at most 0.053. Under all four evaluator configurations, `kimi-k3` remains ahead of `Qwen3.8-27B` on ME and behind it on RE. This is a descriptive sensitivity check rather than a controlled estimate of evaluator effects: the alternative runs use judge-only re-scoring and may also differ from production in sampling and endpoint serving. The stable ordering therefore supports the reported cross-track comparison without establishing evaluator invariance.

D Implementation and Resource Accounting

Execution constraints. Each trajectory has a three-hour wall-clock limit and a 120-turn cap. A wall-clock timeout terminates execution and is recorded as an infrastructure failure. Reaching the turn cap instead triggers a build, render, and evaluation of the final workspace state.

Telemetry and token accounting. Resource totals come from request-level API logs and include retries. For Anthropic routes, token totals sum prompt, completion, cache-creation, and cache-read entries. For OpenAI routes, reported prompt usage already includes cached input. The evaluated trajectories contain 135,370 tool calls. These accounting conventions distinguish actual requests from a count of unique attempted actions.

Trace integrity and final-state evaluation. Tool invocations and returns are paired sequentially within the same assistant scope. This prevents identifiers reused by nested agents from merging unrelated events. Intermediate renders and edits remain available for inspection. Scores use the final candidate: earlier behavior earns credit only if it remains present in the final workspace.

E Representative Benchmark Cases

Figures 5 and 6 connect solver inputs to evaluator checks. Each row shows the runnable page and relevant detail on the left, a summarized request in the center, and three selected acceptance criteria on the right. Full-page views establish the surrounding application; detail views locate the edit or its supporting evidence. The checks cover requested changes, dependent updates, and explicit preservation constraints, illustrating why a plausible local modification may leave part of a task unresolved. These panels illustrate the task interface; quantitative results use the full task pool and complete rubrics, which remain hidden from the solver.

What the ME examples test. The receipt case requires a value to be reconciled across a source document, an editable row, and multiple dependent summaries. The click-audit case tests whether externally supplied counts propagate consistently without disturbing unrelated link metadata. The video-grounded case combines visual-reference transfer with localized preservation: one product card must adopt evidence from the clip while neighboring cards, ordering, price, and rating retain their specified values and positions. Across the three rows, the selected checks follow a common pattern: establish the authoritative evidence, verify every requested dependent update, and test that unrelated content remains stable. The checks assess whether a local patch also satisfies the requested relationships among totals, linked views, and neighboring components.



Figure 5: **Representative ME cases: grounded local edits.** The examples cover cross-source reconciliation, cross-view consistency, and visual-reference grounding. Numbered markers organize the visible evidence; the right column visualizes three selected evaluator targets as value flows, compact charts, and state transitions. The complete rubric may contain additional criteria.

What the RE examples test. These repository-level tasks require behavior to remain coherent beyond a single screenshot. The partner-page case combines navigation restructuring, new content sections, responsive styling, and accessibility requirements. The clip editor couples duration and count limits with recovery, timeline accuracy, export, and keyboard control. The theme case adds persistent color modes, working routes, and a GSAP-driven mega menu while preserving the site's layout and existing motion behavior. Across the three rows, the checks distinguish visible content, executable interaction, and baseline behavior that must remain intact. The checks cover the requested interface together with the state transitions that make its controls and routes usable.

F Dataset Composition and Capability Taxonomy

F.1 Dataset Composition

Table 8 reports task counts, criterion density, reference materials, and starting-artifact metadata. Among RE tasks with capture artifacts, the mean is 5.1 captures per task. Table 9 shows that ME contains 79 single-page HTML applications (79%) and 21 bundled multi-file application trees. In RE, 171 tasks (98.8%) use single-page React applications. A single-page application can still require coordinated edits across several repository files.

F.2 Capability Taxonomy and Multi-Label Coverage

The taxonomy contains three high-level dimensions, nine categories, and thirty fine-grained capabilities (Figure 8). A primary label assigns each task to its main delivery objective. Additional labels

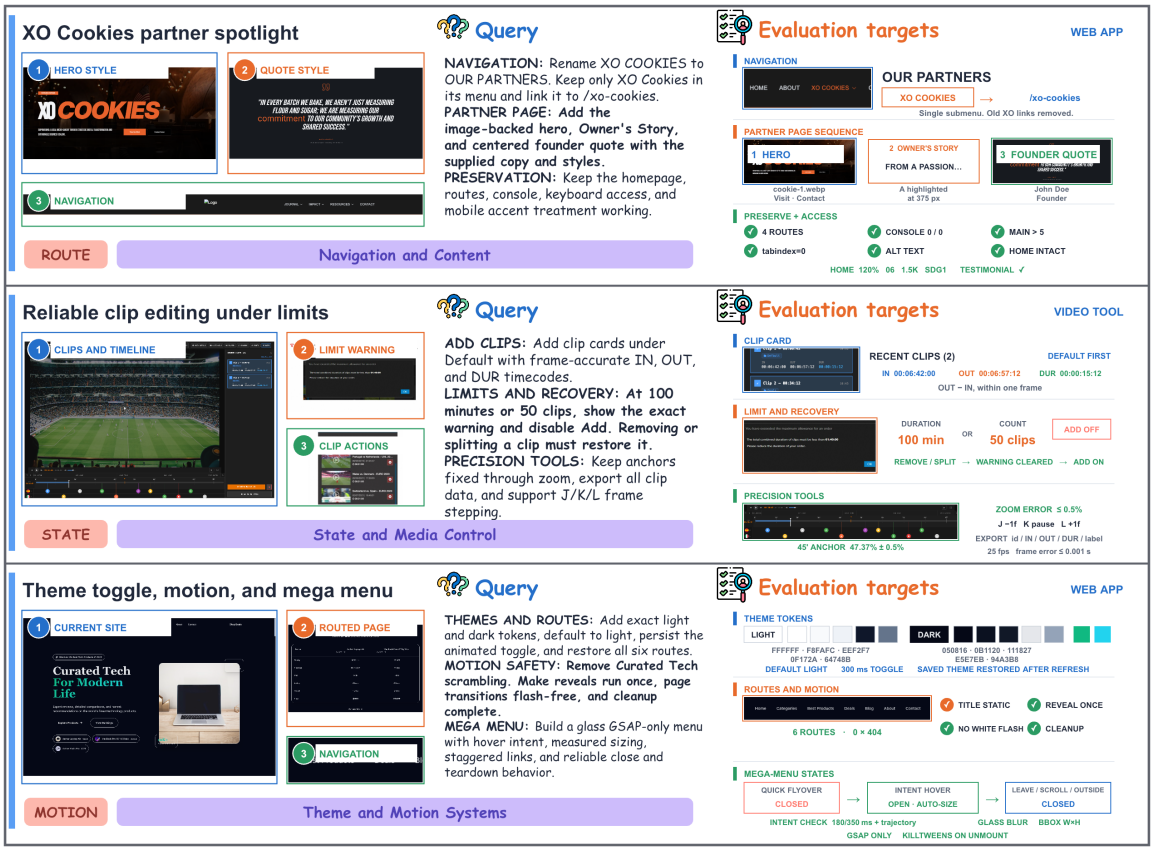

Figure 6: **Representative RE cases: coordinated repository edits.** The examples span navigation and page construction, clip editing under resource limits, and theme-aware motion with a morphing mega menu. The selected targets use route flows, thresholds, token palettes, and interaction states to make the behavior and preservation constraints explicit.

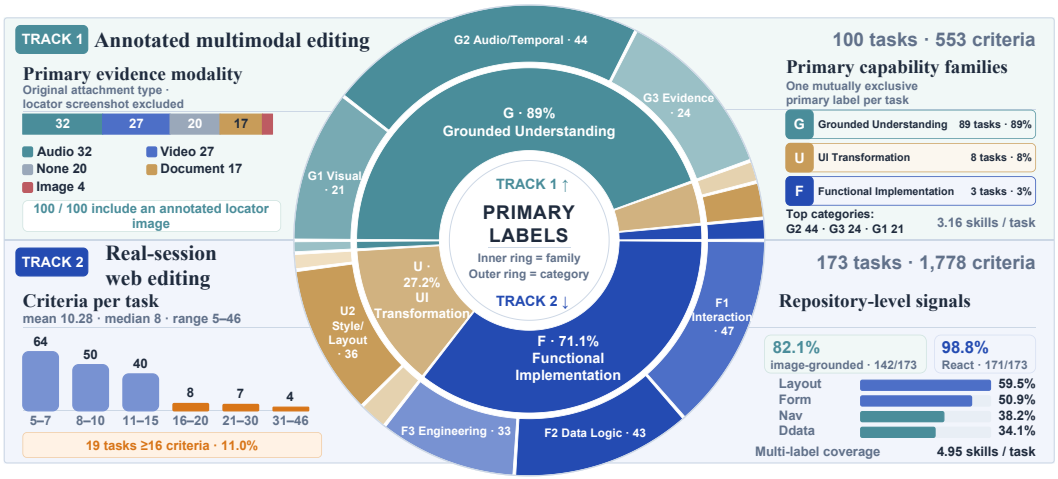

Figure 7: **Two complementary tracks in Co-Design Bench.** Both originate in real user coding sessions. ME groups tasks by primary attachment modality ("None": no additional attachment); RE change types overlap. Bars show shares within each track; criteria per task is the mean. Detailed statistics appear in Table 8.

record other required capabilities, producing 1,172 task–capability associations. Under this annotation scheme, 265 of 273 tasks (97.1%) have multiple capability labels, and 174 (63.7%) span more

Table 8: **Composition of the 273-task benchmark.** Percentages use each track's task count; RE edit types overlap. Mean captures are computed over tasks with captures.

	ME *design edit*	**RE** *whole-app reflow*
Tasks	100	173
Evaluation criteria	553	1,778
per task, mean / median / max	5.5 / 6 / 11	10.3 / 8 / 46
Tasks with attachments / captures	80 (80%)	142 (82%)
Mean reference materials per task	1.35 attachments	5.1 captures
ME: input modality		
None	20 (20%)	–
Image	4 (4%)	–
Video	27 (27%)	–
Audio	32 (32%)	–
Document	17 (17%)	–
RE: edit types		
Structure	–	151 (87%)
Style	–	133 (77%)
Interactive	–	131 (76%)
Content	–	112 (65%)
Multiple edit types	–	166 (96%)
Interaction depth L3–L4	–	101 (58%)
Starting artifact	single page 79, bundled app 21	React 171 (99%)

Table 9: **Starting artifact and attachment modality on ME.** Shading shows task counts; dashes denote zero. Strict ($\bar{R}$) averages task scores across ten endpoints; grey scores represent single-task groups. The bundled subtotal includes both source-tree rows.

	Attachment modality						
Starting artifact	None	Image	Video	Audio	Document	**Total**	**Strict** ↑
single-file HTML page	2	4	25	32	16	79	0.619
bundled application	18	–	–	–	1	19	0.628
REACT source tree	–	–	1	–	–	1	0.440
VUE source tree	–	–	1	–	–	1	0.500
bundled, all three	18	–	2	–	1	21	0.613
All tasks	20	4	27	32	17	100	0.617

than one high-level dimension. Primary-label results partition the task pool; multi-label coverage describes overlapping requirements.

F.3 Category-Level Completion Breakdown

Figure 9 reports mean strict completion for each endpoint and primary category across the full evaluation grid, together with dimension-level means and full-pass rates. The Motion and Adaptation category contains only four tasks, so its scores summarize a particularly small group. Capability labels also align with the two tracks: 89 of 92 Grounded Understanding tasks belong to ME, while 123 of 126 Functional Implementation tasks belong to RE. Category differences therefore combine capability requirements with the editing contexts in which those requirements occur.

G Tool Use and External Perception Models

Cross-track tool activity. The image-inspection counts in Section 5.2 include reads of attachments, extracted video frames, and browser screenshots. Workspace modification counts include `Edit`/`Write` calls. For the long-trajectory analysis, scored rollouts are sorted within each endpoint by turn count, with ties broken by task ID. We select the upper quartile and average its tool shares with equal weight across endpoints. These summaries describe tool activity, without attributing success to a particular tool family.

Media processing. Table 10 reports named media-processing tools found in ME traces. The categories include direct `ffmpeg` calls with specified input streams, OpenCV image or video routines,



Figure 8: **Capability hierarchy and benchmark coverage ($N = 273$).** Primary assignments partition tasks by main delivery goal; Any-label coverage includes auxiliary requirements, allowing overlap across categories. All thirty fine-grained capabilities are represented across the task suite.

and `faster-whisper` transcription. Generated helper scripts count when the trace records their subsequent execution in the shell.

Table 10: **Identifiable media-processing Bash requests by modality.** All final runs of the ten endpoints on the released ME tasks are included. Each cell gives the number of Bash requests and, in parentheses, the percentage of runs with at least one such request. A request counts once per named tool; columns can overlap. Installation, import/version checks, and script preparation alone are excluded; failed processing requests are included.

Modality (tasks)	Runs	`ffmpeg`	OpenCV	`faster-whisper`
No attachment (20)	200	1 (0.5%)	0 (0.0%)	0 (0.0%)
Image (4)	40	0 (0.0%)	0 (0.0%)	0 (0.0%)
Video (27)	270	257 (31.5%)	2,438 (68.5%)	20 (3.3%)
Audio (32)	320	160 (25.0%)	3 (0.3%)	65 (4.4%)
Document (17)	170	0 (0.0%)	96 (3.5%)	0 (0.0%)
All ME (100)	1000	418 (16.6%)	2,537 (19.2%)	85 (2.3%)

G.1 External Pretrained Models

Table 11 gives examples from five solver endpoints that download or execute pretrained perception models from Hugging Face. The audit covers all ten endpoints, including `gpt-6-astra`. Observed outputs include AST audio-class scores, CLAP audio–text similarity scores, Whisper transcripts, and Qwen2-Audio descriptions. The selected rows illustrate available processing strategies; their counts are not prevalence estimates.

Primary capability group	Tasks	gpt-6-astra	opus-5	qwen3.8-max-0902	qwen3.8-27b	kimi-k3	gpt-5.6-sol	opus-4-8	glm-5.3-flash	glm-5.2	grok-4.6	Pooled	Full pass
Grounded Understanding	**92**	**73.8**	**68.6**	**64.1**	**62.2**	**64.4**	**58.4**	**57.5**	**53.7**	**54.2**	**47.1**	**60.4**	**20.2**
Visual understanding	21	67.9	72.4	66.7	67.0	68.7	54.9	63.3	45.7	50.0	40.3	**59.7**	21.0
Audio & temporal underst.	44	74.0	65.3	60.1	55.2	64.4	61.8	55.5	53.2	53.0	47.1	**58.9**	20.2
Evidence integration	27	78.0	70.9	68.7	69.8	61.0	55.8	56.3	60.9	59.4	52.3	**63.3**	19.6
UI Transformation	**55**	**69.2**	**65.2**	**66.8**	**64.2**	**64.0**	**59.6**	**58.0**	**55.9**	**59.3**	**52.3**	**61.5**	**13.6**
Content & assets	10	87.7	82.9	81.2	78.1	78.0	84.1	71.2	67.6	76.1	64.7	**77.2**	28.0
Style & layout	41	66.7	62.7	64.4	61.8	61.3	55.0	56.2	51.4	56.3	50.3	**58.6**	11.5
Motion & adaptation†	4	48.3	46.6	55.7	54.3	56.9	46.0	43.8	72.7	47.7	42.4	**51.4**	0.0
Functional Implementation	**126**	**67.3**	**64.7**	**65.4**	**66.1**	**61.2**	**63.7**	**58.6**	**60.6**	**57.4**	**53.5**	**61.9**	**10.3**
Interaction behaviors	47	67.8	62.8	64.3	61.2	60.9	61.6	54.3	56.4	57.4	51.0	**59.8**	9.1
Data logic	46	66.5	67.3	66.0	68.7	61.9	64.1	61.7	62.9	57.3	61.3	**63.8**	13.9
Engineering integration	33	67.7	64.0	66.2	69.5	60.9	66.0	60.5	63.3	57.3	46.1	**62.2**	7.0
All 273 tasks	**273**	**69.9**	**66.1**	**65.3**	**64.4**	**62.9**	**61.1**	**58.1**	**57.3**	**56.7**	**51.1**	**61.3**	**14.3**



†fewer than five tasks; values greyed

Figure 9: **Mean strict completion ($\bar{R}$) across primary capability categories and model endpoints (%).** Bold rows denote top-level dimensions; sub-rows report constituent categories. Pooled columns report overall average scores across all evaluated pairs, while Full Pass (F) denotes the share of completely resolved tasks. The low-sample Motion and Adaptation category (†, $n = 4$) is displayed in grey.

Table 11: **Hugging Face models in selected final trajectories.** Rows group external models and observed outputs by solver. AST is the Audio Spectrogram Transformer fine-tuned on AudioSet; Whisper variants are invoked through `faster-whisper`.

Solver	External models	Observed evidence
`gpt-6-astra`	AST (AudioSet), CLAP (HTSAT-unfused), Whisper small, Qwen2-Audio-7B-Instruct	AudioSet class scores, audio–text similarity scores, timestamped transcripts, and generated audio descriptions.
`opus-5`	AST (AudioSet), Whisper small	Audio class scores (e.g., Frog 0.91 and Church bell 0.84) and timestamped Chinese transcripts.
`kimi-k3`	AST and YAMNet (ONNX weights)	AST weights downloaded; YAMNet inference returns animal-class scores over time.
`glm-5.3-flash`	CLIP ViT-B/32, DINOv2-small, OWLv2-base (ensemble)	Video-frame scene scores, crop embedding similarities, and object detection boxes.
`glm-5.2`	Qwen2-Audio-7B-Instruct, AST (AudioSet), Whisper base, large-v3	Generated audio descriptions and timestamps, AudioSet class scores, and timestamped transcripts.

Returned inference payloads distinguish successful model use from attempted execution. In one OWLv2 example, a post-processing exception occurs despite a zero shell exit code; a corrected call subsequently returns detection boxes. The traces thus document concrete uses of external perception models through local runtimes, including PyTorch, ONNX Runtime, and `faster-whisper`, without establishing how much these calls improve task completion.

H Cross-Harness Sensitivity

The comparison covers two solvers, `kimi-k3` and `Qwen3.8-27B`, on all 273 tasks (100 ME; 173 RE). We test three harnesses: `codex` 0.142.0, `opencode` 1.18.13, and `qwen-code` 0.21.11 (OpenAI, 2026a; Anomaly, 2026; Qwen, 2026). Prompt templates, acceptance criteria, and judge protocols are shared. This comparison measures how the observed performance of a fixed solver changes with its execution harness.

Table 12: **Completion under three execution harnesses.** Evaluated across all 273 benchmark tasks (ME: 100; RE: 173) with the solver model held fixed while the harness is swapped across `codex`, `opencode`, and `qwen-code`. Cells report average completion rates. The best harness per column is **bold**. Harness rankings invert across solvers on ME (Spearman $\rho = -1.00$), and the performance gap between `kimi-k3` and `Qwen3.8-27B` varies by 10.0 percentage points on ME and 16.2 on RE.

	ME ($n = 100$)		**RE** ($n = 173$)	
Harness	`kimi-k3`	`Qwen3.8-27B`	`kimi-k3`	`Qwen3.8-27B`
`codex`	0.842	0.773	0.649	0.638
`opencode`	0.828	**0.801**	**0.709**	**0.749**
`qwen-code`	**0.849**	0.722	0.682	0.560
Spread (max−min)	0.021	0.079	0.060	0.189
kimi−27B range	$+0.027 \rightarrow +0.127$ (10.0 pp)		$-0.040 \rightarrow +0.122$ (16.2 pp)	

Harness rankings depend on the solver. On ME, the harness ordering reverses between the two solvers (Spearman $\rho = -1.00$). `kimi-k3` scores highest under `qwen-code` and lowest under `opencode`; `Qwen3.8-27B` has the opposite ordering. On RE, the correlation is $+0.50$. Across the three tested harnesses, an ordering obtained for one solver therefore does not reliably predict the ordering for the other.

Relative model scores depend on the harness. The `kimi-k3` minus `Qwen3.8-27B` gap ranges from $+0.027$ to $+0.127$ on ME and from -0.040 to $+0.122$ on RE, spanning 10.0 and 16.2 percentage points. `kimi-k3` remains ahead on ME, while the ordering changes with the harness on RE. These results motivate reporting the model and harness together and holding the harness fixed for a direct model comparison. The comparison changes entire harnesses, so it cannot attribute score differences to individual tool practices.

I Data Curation Details

This appendix specifies the recovery, filtering, and selection procedures summarized in Section 3.2.

Source corpus and notation. The corpus contains 39,325 query records from 26,314 coding sessions, including 25,373 frontend application sessions. Deduplication leaves 37,463 candidates, each anchored to an edit turn, repository, available history, and attachments. In Figure 2, $Q \cdot I \cdot B \cdot C$ is shorthand: $Q = q$, the figure's $I = (X, K)$ combines attachments and context, and the manifest bundled with B specifies environment E.

Environment recovery and validation. Starting applications are recovered from telemetry logs, repository archives, standard scaffolds, or static HTML bundles. Recovery restores code, assets, and available history while recording the original source hash. Validation checks code integrity, dependencies, builds, container launch, and headless Playwright rendering. Candidates requiring inaccessible private APIs, authenticated backends, or proprietary SDKs are excluded. Other faulty repositories undergo P0 scriptable batch repair, P1 agentic repair, or P2 structural pruning, followed by the same validation gate. A validated package contains the baseline, request context, attachments, container manifest, and validation logs.

Deduplication and filtering. Content hashes identify exact duplicates; near-duplicates with tokenized Jaccard similarity above 0.80 are removed. Normalization decodes attachments and replaces local image paths with packaged asset references. An optional OCR and content-inspection module screens confidential references while retaining task specifications. Before ranking, deterministic filters F1–F6 reject environment-blocked turns, non-actionable chatter, unbounded requests, system trajectory artifacts, and non-editing interactions. Candidates must pass every applicable filter.

Candidate ranking. Surviving queries receive a heuristic score combining structural difficulty, verifiability, and historical failure signals:

$$r(q) = d(q) + v(q) + f(q), \qquad d \in [0,5],\ v \in [0,3],\ f \in [0,2]. \tag{4}$$

The structural difficulty score prioritizes coordination requirements: $d = 1$ denotes localized cosmetic edits such as changing one CSS property; $d = 2$–3 denotes dynamic client-side state changes or intra-component reflow; and $d = 4$–5 denotes multi-component refactoring, cross-file data propagation, or complex edge cases. Verifiability depends on the required inspection: static source diffs ($v = 1$), rendered visual DOM properties ($v = 2$), or dynamic post-interaction state transitions ($v = 3$); ambiguous or non-auditable targets receive $v = 0$. Historical failure signals are scored after removing environment-blocked turns: a single-turn resolution receives $f = 0$; two or three query reformulations receive $f = 1$; and four or more repetitive follow-up attempts or an explicit rollback receive $f = 2$.

Session consolidation and rubric authorship. The selector keeps at most three eligible queries ($r(q) > 1$) per session. Their bounding message indices define a closed interval that retains intervening user turns in chronological order. Sessions with question_total_score ≥ 10 proceed to consolidation. For RE, experts refine the request, assign metadata, and author hidden criteria based on the selected user intent. Solver packages contain the task description and working copy; evaluator packages additionally retain the rubrics.

J Evaluation Implementation Details

The evaluator receives the request q, hidden rubric C, baseline B, candidate application W, and execution logs. The procedures below implement the evidence requirements in Section 4.

Sandbox execution and rendering. The candidate is mirrored into an isolated workspace. Fixed render-recovery heuristics address non-blocking environment issues before the evaluator runs the appropriate `vite build` or static HTML pipeline, starts the application server, and launches headless Chrome. Build, server, and browser outcomes become part of the retained execution evidence.

Evidence extraction and primary grading. Evidence includes source diffs, DOM trees, body text, viewport screenshots, and source-to-entry reachability logs. When interaction checks are active, Playwright records action sequences and network telemetry. The primary evaluator applies the hidden rubric to this evidence with the baseline available for comparison. All primary passes share an evaluator configuration and Playwright automation; a fixed configuration does not make LLM generations deterministic. Output normalization preserves the original criterion count $|C_t| = n_t$.

Evidence requirements by criterion type. Reachable source and DOM state can support static criteria. Visual claims require a successful render and screenshot evidence; interaction claims also require recorded user actions and the resulting state transition. Internal state or a handler in source code cannot substitute for an executed interaction. Failed builds or missing DOM nodes invalidate render-dependent claims, and missing required evidence cannot be replaced with a weaker proxy.

Scoring conventions. Strict Completion counts only PASS; Graded Completion also gives half credit to PARTIAL, ensuring $0 \leq R_t \leq G_t \leq 1$. A zero strict score can therefore coexist with partial credit. Pooled task scores average the model–task grid, while criterion-verdict shares pool individual judgments. Non-editing conversational and diagnostic auxiliary records are excluded from implementation-scoring denominators. Raw and environment-adjusted scores are retained separately so that attribution does not overwrite the original judgments.

Human review of rubric scoring. Two frontend experts checked the judge model's rubric scores for a total of 200 task–model outputs. They examined the task requirements, evaluation rubric, and generated interface, including browser interactions when required, to check whether the criterion-level PASS/PARTIAL/FAIL verdicts reflected the output's satisfaction of the corresponding requirements.

Audit trail and reproducibility. Each record links the request and rubric to the baseline, final workspace, build outcome, evidence, primary verdicts, and refutation trace. This chain supports criterion-level inspection and score re-aggregation without revealing hidden rubrics to the solver. Stored outputs document the sampled solver and evaluator generations; an endpoint name alone is insufficient to reproduce those generations exactly.

J.1 CONDITIONAL FAILURE ATTRIBUTION

Attribution is triggered for low-scoring cases ($G_t < 0.8$) when an independently rendered baseline passes the startability gate. A parallel solvability probe attempts a minimal patch in an isolated sandbox, stubbing inaccessible network endpoints while preserving core business logic. Unfulfilled criteria are attributed to model limitations, environmental blockage, mixed causes, or initial-artifact defects. These diagnoses leave raw scores and denominators unchanged. Supplementary environment-adjusted scores grant attribution credit only when infrastructure failures are established.