VIRE-BENCH: BENCHMARKING REPOSITORY-LEVEL FRONTEND CODE EDITING FROM VISUAL FEEDBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontend editing requests often concern visible content, appearance, and spatial relationships. Annotated screenshots ground these requests directly in the rendered interface, allowing users to communicate changes to coding agents without identifying source files or components. Agents must then infer the intended edit, locate the relevant repository code, and implement the change while preserving unrelated content and appearance. We introduce ViRE-Bench, a benchmark of 1,000 repository-level frontend code-editing tasks drawn from 202 real-world HTML, React, Vue, and Angular repositories. The benchmark spans eight edit operations, covers both instance-level and shared-component edits that propagate across pages, and evaluates submission validity and intent satisfaction alongside rendered measures of target fidelity and preservation. Among six models evaluated under a common single-turn protocol, the best-performing model achieves an overall benchmark score of 67.29 and an intent satisfaction rate of 70.2%. Component movement and resizing remain challenging for all six models. Controlled diagnostic experiments show that explicit editing intent yields larger gains than source-location information alone across all three tested models, with two models reaching approximately 95% intent satisfaction under explicit intent. These findings indicate that deriving an actionable editing specification from visual feedback is a primary source of difficulty, while the benefits of source-location assistance vary across models.

1 INTRODUCTION

Frontend code defines an interface's content, appearance, and layout, which users experience through the rendered page. This visual context provides a natural setting for users to indicate desired changes. Users may know how large a button should be or where a card should move without knowing the corresponding pixel dimensions or displacement. Annotated screenshots let users mark targets, draw desired boundaries or positions, and add brief notes directly on the interface. This reduces the need to translate spatial intent into numerical parameters or detailed prose. Users need not know which source files or components implement the interface or how to modify them.

Coding agents increasingly perform repository-level development tasks, providing a means to translate these visual requests into code changes (Yang et al., 2024; Zhang et al., 2024). An agent must infer the target, operation, and parameters from the annotations, locate the relevant implementation, and edit existing components and styles while preserving unrelated content. Changes to shared interface elements may also need to propagate consistently across pages. Figure 1 illustrates how users express editing requests visually and how coding agents translate them into repository changes.

Existing benchmarks examine screenshot-based interface reconstruction (Si et al., 2025; Gui et al., 2025), frontend code editing with explicit textual instructions (Xiao et al., 2025; Awal et al., 2025), and multi-turn development and localized repair (Wu et al., 2026; Li et al., 2026). However, systematic evaluation of localized repository edits specified through annotated screenshots remains limited, particularly when changes must preserve unrelated content and remain consistent across pages. We ask whether coding agents can interpret such requests and implement them in real, pre-existing multi-file repositories. Task-specific requirements come from the annotated screenshot, without a separate textual editing instruction, a complete target rendering, or explicit source-location hints.

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.

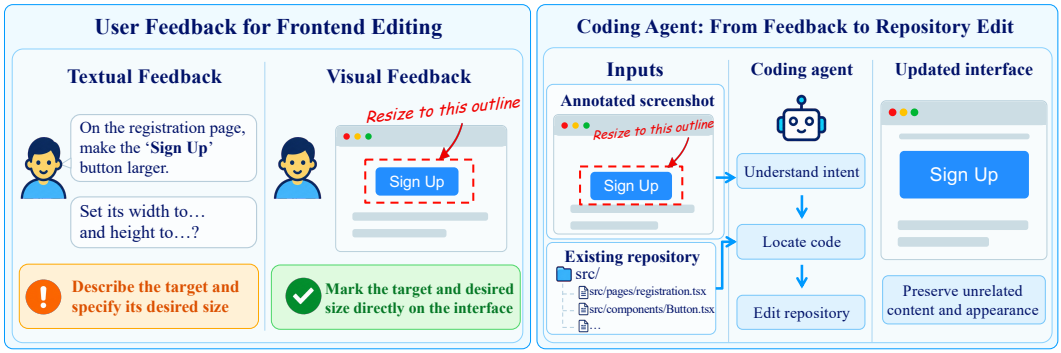


Figure 1: Visual-feedback-driven repository editing. Left: expressing desired changes through text or visual annotations. Right: translating annotated requests into repository edits.

We introduce ViRE-Bench, a benchmark of 1,000 visual-feedback-driven frontend code-editing tasks drawn from 202 real-world repositories across HTML, React, Vue, and Angular. It covers eight edit operations, including both instance-level edits and shared-component edits across pages. Under a common single-turn protocol, we assess submission validity and intent satisfaction, alongside target fidelity and preservation of unrelated content across relevant pages.

Across six models, the best-performing model achieves a mean task score of 67.29 and an intent satisfaction rate of 70.2%. Text changes are the most reliable operation, whereas component movement and resizing remain consistently difficult across models. Diagnostic experiments with three models on a fixed 320-task subset provide explicit editing intent, source-location information, or both. Explicit intent yields larger gains in intent satisfaction than source-location information alone for all three models. Gemini-3.7-Flash and GPT-5.6-Sol reach approximately 95% intent satisfaction with explicit intent, while Qwen3-VL-235B-A22B-Thinking reaches 74.4% with both supplements. These results indicate that deriving an actionable editing specification from visual feedback is a primary source of difficulty. Additional gains from source-location information vary across models.

Our contributions are: (1) a **repository-level frontend code-editing benchmark** spanning four frontend technologies and eight edit operations; (2) a **construction pipeline** linking structured edit specifications to programmatically generated reference implementations and visual feedback; (3) an **evaluation protocol** separating validity, intent satisfaction, target fidelity, and preservation across pages; and (4) **six-model experiments and diagnostic interventions** examining the effects of explicit editing intent and source-location information.

2 Related Work

Visual-to-code generation. Design2Code benchmarks webpage reconstruction from screenshots (Si et al., 2025), while WebCode2M provides large-scale screenshot–code pairs (Gui et al., 2025). MRWeb extends generation to multi-page websites using explicit resource lists (Wan et al., 2024). 1D-Bench evaluates React code generation and iterative refinement with execution feedback (Xu et al., 2026). Vision2Web spans static interface generation, interactive multi-page reproduction, and full-stack website development (He et al., 2026). These studies focus on implementing interfaces from visual designs. In ViRE-Bench, an annotated screenshot instead specifies a localized edit to an existing interface without providing a complete screenshot of the intended result.

Visual feedback and frontend code editing. DesignBench and WebMMU evaluate frontend code editing from screenshots, source code, and explicit textual modification requests (Xiao et al., 2025; Awal et al., 2025). In ViRE-Bench, annotated screenshots convey task-specific requests, requiring agents to infer the target, operation, and parameters without separate textual editing instructions. Sketch2Code studies interactive prototyping from low-fidelity sketches with simulated-user feedback and model-initiated clarification (Li et al., 2025). WebGen-Agent refines generated websites through screenshot and GUI-testing feedback (Lu et al., 2025). Our tasks instead start from supplied implementations and require changes within existing code. MT-Web2Code constructs multi-turn

repair tasks through synthetic corruption, using annotated screenshots to mark repair regions and reference image crops to guide regional reconstruction (Li et al., 2026). In ViRE-Bench, annotations specify desired changes to the current interface, rather than directing recovery of a state preceding synthetic corruption. FronTalk studies conversational website development with textual or visual feedback adapted to model-generated implementations (Wu et al., 2026). ViRE-Bench instead grounds visual editing requests in real, pre-existing multi-file repositories, requiring agents to locate and modify the relevant implementations. Its single-turn tasks cover both instance-level changes and shared-component edits across pages.

Repository-level multimodal editing. SWE-bench evaluates repository-level issue resolution using automated tests (Jimenez et al., 2024). SWE-bench Multimodal extends this setting to visual software domains, incorporating images in problem statements or unit tests (Yang et al., 2025). GALA aligns visual interface structures with repository code for bug localization (Liu et al., 2026), while VisualRepair combines image-type-aware tool use with adaptive region focusing for visual issue repair (Xiao et al., 2026). ViRE-Bench complements this line of work by evaluating frontend editing requests expressed directly through annotated screenshots. Users specify desired interface changes, and agents interpret these requests to locate and modify the relevant code. Evaluation separates operation-specific intent satisfaction on the primary page from target fidelity and preservation across relevant rendered pages.

3 ViRE-Bench

3.1 Task Definition

ViRE-Bench evaluates agents' ability to edit real-world frontend repositories from visual feedback. Given an initial repository R and a feedback image I_{fb}, an agent must interpret the requested edit, locate the relevant source code, and produce a modified repository R' that implements the change. The feedback image is a screenshot of the current interface annotated with visual marks and optional textual notes. All tasks share a fixed instruction (Appendix A), while the feedback image conveys task-specific targets and editing requirements.

We consider eight edit operations: *text change*, *style change*, *resizing*, *alignment change*, *component movement*, *component reordering*, *component addition*, and *component removal*. Each task involves one operation under a single-turn protocol: agents may make multiple tool calls to inspect and edit the repository but receive no follow-up user feedback. Appendix D illustrates visual feedback for all eight edit operations.

The objective is to implement the requested edit while preserving unrelated content and appearance. The modified project must remain valid and render successfully. Instance-level tasks modify a selected interface instance, whereas shared-component tasks require consistent changes to corresponding instances across affected pages. Agents must infer the intended scope from the visual feedback and repository structure, without explicit scope labels or affected-route lists.

3.2 Benchmark Construction

We construct ViRE-Bench in five stages: repository collection and filtering, resource freezing and project curation, edit intent generation, reference implementation and visual feedback generation, and task validation and curation. Figure 2 summarizes the construction pipeline and the subsequent agent evaluation.

Repository collection and filtering. We collect candidate GitHub repositories spanning HTML, React, Vue, and Angular. For each repository, we pin a revision and identify the project root, entry points, and build configuration primarily through rule-based analysis. When this analysis is inconclusive, we use Codex (OpenAI, 2025) with GPT-5.6-Luna (OpenAI, 2026) to inspect repository documentation and configuration files and determine the required setup. We build projects as needed and retain repositories with at least one route that renders nonempty content without page-level errors or missing critical local resources. Appendix B reports candidate counts, Codex fallback usage, and filtering outcomes.

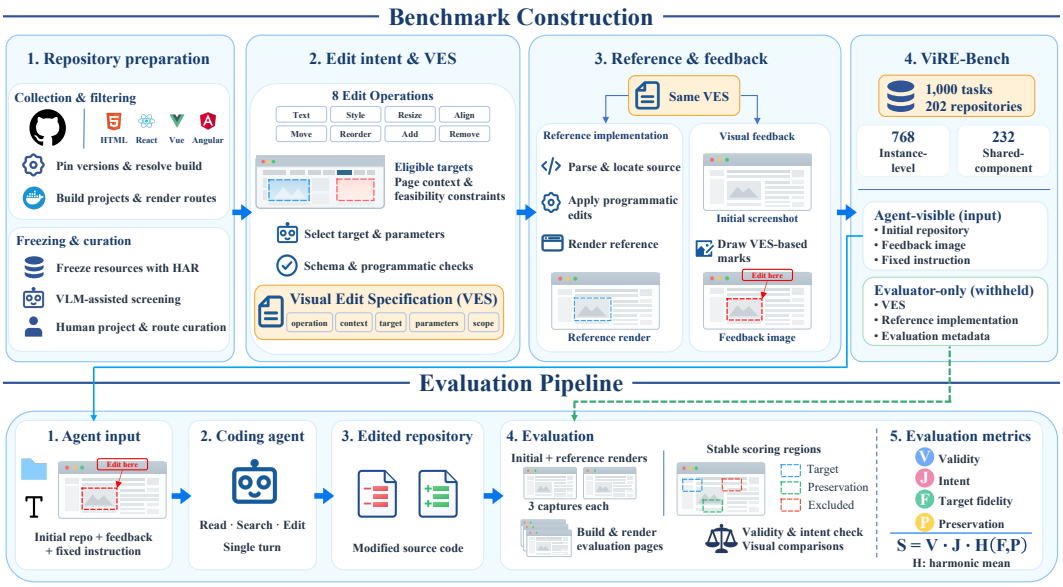


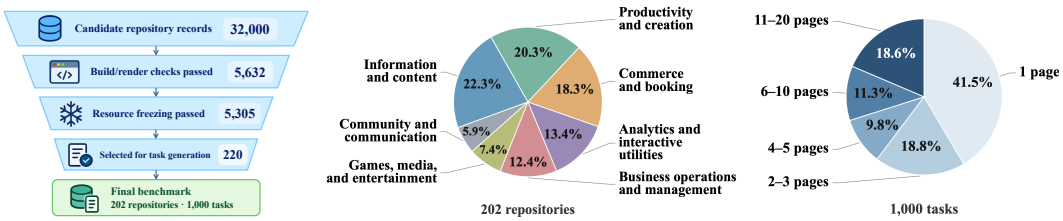
Figure 2: Overview of ViRE-Bench. Top: construction of 1,000 tasks from 202 repositories, with separate agent inputs and evaluator-only artifacts. Bottom: single-turn repository editing followed by validity and intent checks and multi-page evaluation of target fidelity and preservation.

Resource freezing and project curation. To reduce rendering variability, we record external resources in HTTP Archive (HAR) files and replay them with external network access disabled. We use Gemini-3.7-Flash (Google DeepMind, 2026) to assess visual quality, interface completeness, and editing suitability from representative screenshots and to propose application categories. Guided by these assessments, human reviewers select diverse projects and routes, remove duplicates, and finalize the application categories.

Edit intent generation. For each task, we fix the edit operation and identify eligible targets using operation-specific constraints, with model-assisted shortlisting when needed. We use Gemini-3.7-Flash to select a target from candidates highlighted in the screenshot and propose edit parameters, aiming for changes appropriate to the page context. After validation and normalization, we encode the operation, target, parameters, and page context in a *Visual Edit Specification* (VES). We then programmatically infer cross-page scope and record it in the VES. Scope inference uses source-node matching for React, Vue, and Angular, and rule-based matching within common interface regions, such as navigation bars and footers, for HTML. Appendix E details the generation prompts and requirements.

Reference implementation and visual feedback generation. We generate the reference implementation and visual feedback from the same VES so that both reflect the same editing intent. Technology-specific parsers and temporary element identifiers map rendered targets to source locations. We then apply operation-specific programmatic edits following the parameters and scope recorded in the VES. We validate the rendered changes and, where needed, record the resulting geometry in the VES to guide feedback generation. Using this specification, we annotate the initial screenshot with visual marks and optional textual notes that convey the implemented change. Appendix F provides implementation details and an example linking a VES to its source edit, visual feedback, and rendered result.

Task validation and curation. To ensure benchmark quality, we combine automated checks, developer review, and a separate human evaluation of the final dataset. Reference edits must produce visible changes and pass operation-specific checks; duplicate tasks are removed. Benchmark developers then review request clarity, feedback–reference consistency, and contextual appropriateness, discarding ambiguous or inconsistent tasks. Three master's students in computer science who were not involved in the project subsequently evaluate 96 tasks sampled from the final benchmark. Re-



(a) Repository selection (b) Application categories (c) Evaluation pages per task

Figure 3: Dataset selection, composition, and evaluation scope. (a) Repository selection; funnel widths are schematic. (b) Application-category distribution across 202 repositories. (c) Distribution of evaluation-page counts across 1,000 tasks. Evaluation pages include those checked for unintended changes outside the edit scope.

quest clarity, contextual appropriateness, reference correctness, and preservation of unrelated content each receive majority approval for 95/96 tasks. Appendix I details the evaluation protocol and results.

3.3 Dataset Statistics

Our collection and curation pipeline starts with 32,000 candidate records representing 31,767 unique repositories and yields 1,000 tasks across 202 repositories. Each frontend technology contributes 250 tasks covering all eight edit operations, with 92–168 tasks per operation across technologies (Appendix C). The benchmark comprises 768 instance-level and 232 shared-component tasks. Figure 3 summarizes repository selection, application categories, and evaluation pages per task.

4 Evaluation Protocol

4.1 Agent Setup

We evaluate all models using OpenCode v1.18.27 (Anomaly, 2026) with a common agent configuration. For each task, the agent receives the fixed instruction (Appendix A) and feedback image in an isolated workspace containing the task repository. We define the primary page as the page shown in the feedback image. Agents may read, search, and edit files, but cannot execute shell commands, install dependencies, or run servers, builds, or tests. This setup standardizes tool access across models and supports comparison of their editing capabilities without execution feedback. Each run is limited to 64 agent steps, 40 tool calls, and 20 minutes. The evaluator then builds and renders the final workspace for scoring. Figure 2 (bottom) summarizes the evaluation pipeline.

4.2 Rendering and Reference Stability

We render each task's predefined evaluation pages for the initial, reference, and agent-modified projects under matched browser and viewport settings, replaying frozen external resources. These pages include the primary page, other affected pages, and pages checked for unintended changes outside the edit scope.

Animations and other dynamic content unrelated to the requested edit can confound visual scoring. To identify this variability, we capture each evaluation page three times for both the initial and reference versions. Regions identified as unstable in either version are excluded from visual scoring. Stable differences between the initial and reference versions define the target region, while the remaining stable areas form the preservation region. Both scoring regions are derived solely from the initial and reference captures, independently of agent outputs. Appendix G illustrates the procedure and provides implementation details.

4.3 Evaluation Metrics

Our evaluation distinguishes submission validity, satisfaction of the user's request, and implementation quality. Validity requires source-code modifications that pass the required checks, including syntax validation. Intent satisfaction determines whether the edit fulfills the user's request. Target fidelity measures visual agreement with the reference in target regions, while preservation measures how well unrelated content and appearance are retained.

Validity and intent satisfaction. Validity $V \in \{0, 1\}$ indicates whether a submission modifies the source code and passes project safety, syntax, build, and rendering checks. Intent satisfaction $J \in \{0, 1\}$ is assessed through operation-specific checks of rendered content, styles, structure, and geometry on the primary page. These checks determine whether the requested operation has been implemented without requiring exact visual agreement with the reference. Appendix H details the matching rules, thresholds, and supported alternative implementations.

Target fidelity and preservation. Target fidelity F measures visual similarity between the agent and reference renders in target regions. Preservation P measures visual similarity between the agent and initial renders in preservation regions. Both metrics combine SSIM and LPIPS to capture complementary aspects of visual similarity. SSIM compares local luminance, contrast, and structure (Wang et al., 2004), whereas LPIPS measures perceptual differences using deep features calibrated against human judgments (Zhang et al., 2018).

For images A and B, we compute both metrics within a scoring region M. For LPIPS, we replace pixels outside M in A with the corresponding pixels in B, giving both inputs the same surrounding content. Let $q_{\mathrm{SSIM}}(A, B; M)$ denote the mean SSIM score within M, clipped to $[0, 1]$. Similarly, let $d_{\mathrm{LPIPS},M}(A, B)$ denote the mean LPIPS distance within M. We convert LPIPS distance to similarity using an exponential transformation and average the two scores:

$$Q(A, B; M) = \frac{q_{\mathrm{SSIM}}(A, B; M) + \exp(-d_{\mathrm{LPIPS},M}(A, B))}{2}. \tag{1}$$

The resulting score lies in $[0, 1]$, with higher values indicating greater visual similarity.

We apply Q to the corresponding image pairs and regions to obtain page-level scores for F and P. Each metric assigns equal weight to the primary-page score and the mean over additional pages. These comprise other affected pages for F and all other evaluation pages for P. When no additional pages contribute to a metric, we use the primary-page score directly.

Overall score. We use the harmonic mean of target fidelity and preservation to limit the overall score when either metric is low. We then apply validity and intent satisfaction as binary gates:

$$S = V \cdot J \cdot \begin{cases} \dfrac{2FP}{F+P}, & F + P > 0, \\ 0, & F = P = 0. \end{cases} \tag{2}$$

A submission receives a score of zero if either gate fails. We report the mean of S over all benchmark tasks on a 0–100 scale, alongside the individual metrics.

Human validation. To validate our automatic evaluation, we compare its results with independent assessments by three annotators covering 192 submissions from 96 tasks. Automatic intent checks agree with human majority judgments on 166 of 173 valid submissions with definite majority judgments (96.0%). Automatic overall scores correlate strongly with median human quality ratings across all 192 submissions (Spearman's $\rho = 0.892$; 95% CI: [0.847, 0.933]). This agreement with human assessments supports the use of our intent checks and overall scores as measures of editing performance. Appendix I details the annotation protocol and analysis.

5 Experiments

5.1 Experimental Setup

We evaluate six models on all 1,000 tasks in ViRE-Bench: Gemini-3.7-Flash (Google DeepMind, 2026), GLM-5.3-Flash (GLM, 2026), GPT-5.6-Sol (OpenAI, 2026), DeepSeek-V4.1-

Flash (DeepSeek-AI, 2026), Claude-Opus-5 (Anthropic, 2026), and Qwen3-VL-235B-A22B-Thinking (Qwen Team, 2025). All runs follow the common agent and evaluation protocol described in Section 4. We set reasoning effort to high for all models except Qwen3-VL-235B-A22B-Thinking, which uses its native thinking mode without a configurable reasoning-effort level. For each model, we report validity and intent satisfaction rates and the mean task score over all 1,000 tasks. We average target fidelity and preservation only over each model's submissions satisfying $V = J = 1$. We compute all aggregate statistics and differences from unrounded values, applying rounding only for presentation.

5.2 Main Results

Table 1: Overall results on ViRE-Bench. Validity and intent are percentages; fidelity, preservation, and score are on a 0–100 scale.

Model	Validity	Intent	Fidelity†	Preservation†	Score
Gemini-3.7-Flash	97.8	**70.2**	93.74	99.42	**67.29**
GLM-5.3-Flash	96.7	60.0	93.92	99.52	57.63
GPT-5.6-Sol	**99.2**	46.1	94.92	99.57	44.58
DeepSeek-V4.1-Flash	76.9	42.8	94.04	99.73	41.19
Claude-Opus-5	96.1	33.3	94.01	99.81	32.02
Qwen3-VL-235B-A22B-Thinking	85.2	21.0	95.89	99.82	20.47

† Conditional means over submissions satisfying $V = J = 1$; qualifying subsets differ across models.

Gemini-3.7-Flash achieves the highest intent satisfaction rate of 70.2% and mean task score of 67.29 (Table 1). GLM-5.3-Flash ranks second on both metrics, reaching 60.0% and 57.63, respectively. Even the best-performing model fails the intent checks on 298 of the 1,000 tasks.

Validity rates substantially exceed intent satisfaction rates across all six models. GPT-5.6-Sol achieves the highest validity rate of 99.2%, yet satisfies the requested intent on only 46.1% of tasks. This gap illustrates that passing validity checks does not ensure implementation of the requested edit. Appendix J summarizes validity failures and budget-exhaustion status among invalid submissions across all six models.

Among submissions satisfying $V = J = 1$, mean target fidelity ranges from 93.74 to 95.89 and preservation from 99.42 to 99.82. With these high conditional visual scores, mean task scores closely track intent satisfaction under the gated scoring scheme. However, the operation-level analysis below reveals differences in visual quality even among submissions that pass both gates.

5.3 Performance Analysis

Edit operation. Overall scores vary substantially across edit operations (Figure 4). All models achieve their highest scores on text change, with values ranging from 87.4 to 97.8. Spatial edits are more challenging, particularly component movement and resizing, for which the highest scores are only 15.1 and 27.4, respectively. Alignment and component reordering show large differences across models. Gemini-3.7-Flash achieves scores of 70.0 and 72.9, whereas GPT-5.6-Sol reaches 37.2 and 11.3, respectively.

Conditional visual scores reveal difficulties with movement and resizing even among submissions satisfying $V = J = 1$ (Appendix K). For Gemini-3.7-Flash, mean target fidelity is only 56.7 for component movement and 75.5 for resizing. The corresponding mean preservation scores are 96.5 and 96.2, respectively. Passing the intent check therefore does not guarantee close visual agreement with the reference in the target region. These results support evaluating visual implementation quality alongside intent satisfaction.

Frontend technology. Overall scores vary much less across frontend technologies than across edit operations (Table 2). For each model, scores across HTML, React, Vue, and Angular span less than 6 points on the 0–100 scale. Model rankings remain unchanged across the four technologies, and no single technology yields the highest or lowest score for all models.

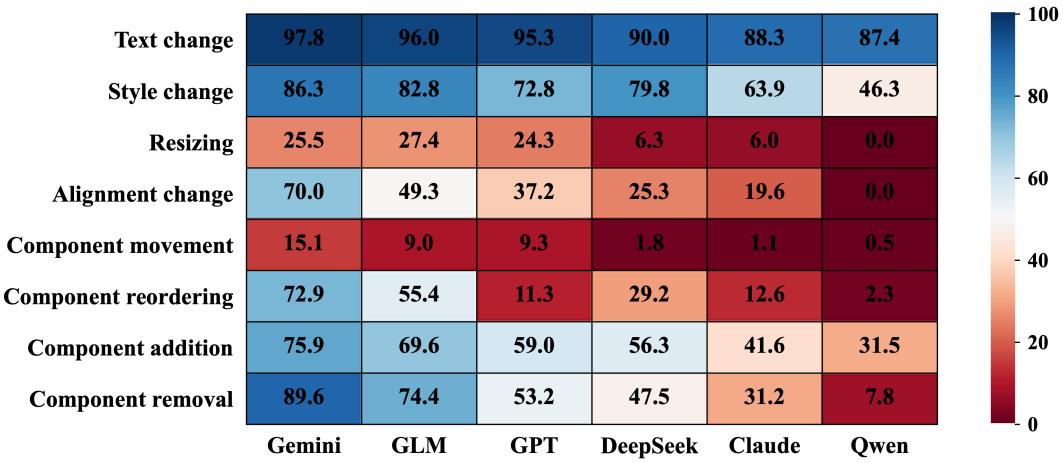


Figure 4: Mean overall scores by edit operation and model on a 0–100 scale. Each cell averages all tasks for the corresponding operation, with submissions failing either validity or intent satisfaction assigned zero. Blue indicates higher scores and red indicates lower scores. Model names are abbreviated from Table 1.

Table 2: Mean overall scores by frontend technology on a 0–100 scale. Each technology has 250 tasks; submissions failing either validity or intent satisfaction receive zero.

Technology	Gemini	GLM	GPT	DeepSeek	Claude	Qwen
HTML	**64.91**	57.04	47.33	43.31	29.24	21.45
React	**69.26**	57.46	44.36	43.74	33.44	22.34
Vue	**68.03**	56.43	43.24	38.53	30.56	16.72
Angular	**66.95**	59.59	43.41	39.17	34.85	21.38
Overall	**67.29**	57.63	44.58	41.19	32.02	20.47

Cross-page consistency. Visual feedback may mark a shared interface element, such as a navigation bar, on a single page without specifying all pages requiring the change. We therefore assess whether edits passing the primary-page intent check are implemented consistently across other affected pages. For shared-component submissions satisfying $V = J = 1$, we compare primary-page target fidelity with the within-task mean over other affected pages. Across the six models, mean target fidelity is 3.2–5.4 points lower on other affected pages than on the primary page (Appendix L).

Patch inspection reveals edits confined to individual HTML pages or page-specific component inputs and styles, leaving corresponding elements unchanged elsewhere. Other cases bind navigation styles to the active state rather than the intended item. These cases highlight the need to infer an edit's scope from repository structure and implement it consistently across affected pages. They also show why passing the primary-page intent check alone is insufficient to establish cross-page consistency.

5.4 Diagnosing Editing Difficulties

Editing from visual feedback requires interpreting the requested edit, locating the relevant source code, and implementing the change. To investigate where difficulties arise, we compare editing performance with explicit editing intent, source-location information, or both. We evaluate Gemini-3.7-Flash, GPT-5.6-Sol, and Qwen3-VL-235B-A22B-Thinking on a fixed 320-task subset balanced across edit operations and frontend technologies. Appendix M describes the subset construction and diagnostic experimental setup.

All variants retain the initial repository, feedback image, and fixed instruction; **Standard input** uses these inputs without additional information. We derive the intent and source-location supplements from the VES. **+ Location** uses recorded target identities to locate the corresponding source code in the initial repository and provides file paths, line numbers, and short code excerpts. **+ Intent**

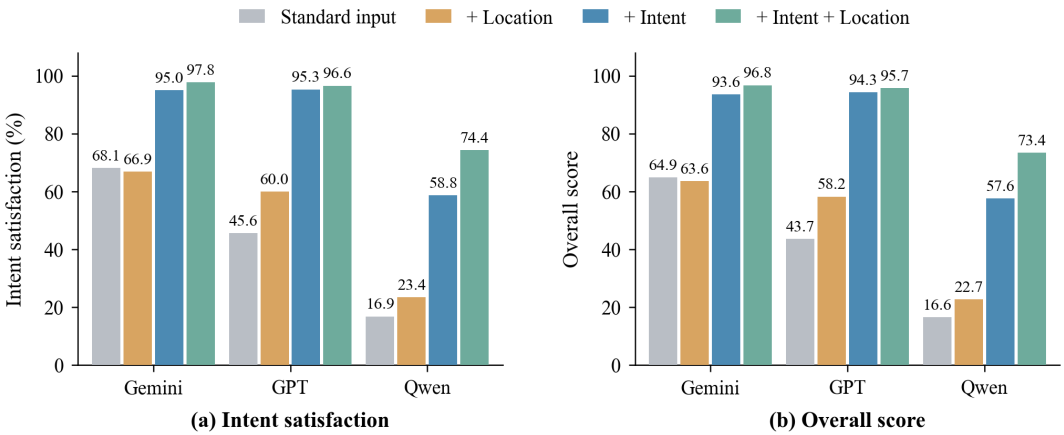


Figure 5: Diagnostic interventions on the fixed 320-task subset: (a) intent satisfaction (%) and (b) mean overall score (0–100). Both metrics include all 320 tasks; submissions failing either gate receive zero overall score.

specifies the target, requested operation, and edit parameters in text, without source locations. **+ Intent + Location** provides both supplements. Neither supplement includes the reference patch or instructions for implementing the edit.

Explicit editing intent yields larger gains than source-location information alone in both intent satisfaction and overall score across all three models (Figure 5). Relative to **Standard input**, **+ Intent** increases intent satisfaction by 26.9–49.7 percentage points, compared with changes ranging from -1.3 to 14.4 points under **+ Location**. These results indicate that deriving an actionable editing specification from visual feedback is a primary source of difficulty.

With explicit intent, Gemini and GPT achieve near-ceiling intent satisfaction rates of 95.0% and 95.3%, respectively, with overall scores above 93. Adding source-location information yields only modest further gains in both metrics for these models.

Qwen benefits more substantially from source-location information after intent is specified: intent satisfaction rises from 58.8% to 74.4%, and the overall score increases from 57.60 to 73.44. The additional benefit of source-location assistance therefore varies across models.

Appendix N presents one successful edit and five failure cases illustrating how intent checks, target fidelity, preservation, and multi-page evaluation capture distinct editing failures. A comparison across input conditions further illustrates how explicit intent can help correct an unsuccessful edit.

6 Conclusion and Discussion

We introduced ViRE-Bench, a benchmark of 1,000 visual-feedback-driven editing tasks across 202 frontend repositories. Its evaluation combines validity and intent checks with multi-page measures of target fidelity and preservation. Across six models, component movement, resizing, and consistent editing across pages remain challenging. Diagnostic experiments show that explicit editing intent yields larger gains in intent satisfaction and overall score than source-location information alone. These findings indicate that deriving an actionable editing specification from visual feedback is a primary source of difficulty. ViRE-Bench supports the development of agents that implement visual requests accurately and consistently while preserving unrelated content and appearance.

Limitations. ViRE-Bench focuses on eight frontend edit operations and does not cover multi-step feature implementation or backend logic. Feedback annotations are generated from structured editing specifications; performance on independently collected human feedback remains untested. Our experiments follow a single-turn protocol without agent-side execution or intermediate rendering. The effects of user interaction and execution feedback remain to be investigated. Evaluation assesses edits across predefined page states but does not test interactive behavior.

AI USE STATEMENT

We used generative AI models during benchmark construction and as evaluated agents. A Codex-based agent identified repository configurations, and vision-language models supported project screening and categorization. Gemini-3.7-Flash selected editing targets and parameters from annotated screenshots; these selections were then processed by programmatic editors and annotation routines to generate reference implementations and feedback images. All generated tasks underwent automated validation followed by manual review to confirm request clarity, reference correctness, and contextual appropriateness. Under the experimental protocol described in Section 4, we used APIs to obtain code-editing submissions from six models: Gemini-3.7-Flash, GLM-5.3-Flash, GPT-5.6-Sol, DeepSeek-V4.1-Flash, Claude-Opus-5, and Qwen3-VL-235B-A22B-Thinking.

We additionally used AI tools to assist with code development and debugging, to provide feedback on experimental design and result interpretation, and to support manuscript drafting and editing. The human authors led the conception of the core research ideas, designed the experiments, curated the benchmark, and prepared the manuscript. The authors retain full responsibility for all research decisions and for the accuracy, originality, and integrity of this work.

ETHICS STATEMENT

ViRE-Bench is designed for academic research on visual-feedback-driven code editing using publicly accessible frontend repositories. We will release repository URLs, fixed commit identifiers, task specifications, and tools for reconstruction and evaluation. We will not redistribute third-party source repositories or external-resource recordings. Users will obtain the source projects and external resources directly from their original sources under the applicable licenses and terms.

REPRODUCIBILITY STATEMENT

Section 3 describes benchmark construction, including project preparation, editing specifications, reference implementations, and visual feedback. Section 4 details the agent protocol, execution budgets, rendering procedure, and evaluation metrics. Section 5 describes the evaluated models and experimental settings. The appendices provide the fixed agent instruction, operation-specific intent checks, region-partitioning procedures, and diagnostic input construction.

We will release repository URLs, fixed commit identifiers, and task specifications, together with reconstruction code and environment configurations. These materials will support reference-edit generation, feedback annotation, rendering, and evaluation without repeating model-based intent selection. Reconstruction scripts will retrieve external resources from their original sources using resource manifests. Before local replay, the scripts will verify resource content hashes and replay metadata against the recorded versions.

REFERENCES

Anomaly. OpenCode: Version v1.18.27. Software release, 2026. URL https://github.com/anomalyco/opencode/releases/tag/v1.18.27.

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

Rabiul Awal, Mahsa Massoud, Aarash Feizi, Zichao Li, Suyuchen Wang, Christopher Pal, Aishwarya Agrawal, David Vázquez, Siva Reddy, Juan A. Rodríguez, Perouz Taslakian, Spandana Gella, and Sai Rajeswar. Webmmu: A benchmark for multimodal multilingual website understanding and code generation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 25118–25145. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1276. URL https://doi.org/10.18653/v1/2025.emnlp-main.1276.

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026. URL https://arxiv.org/abs/2609.19969.

GLM. GLM-5: from vibe coding to agentic engineering. *CoRR*, abs/2602.15763, 2026. doi: 10.48550/ARXIV.2602.15763. URL https://doi.org/10.48550/arXiv.2602.15763.

Google DeepMind. Gemini 3.7 Flash. Model card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-7-flash/.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan Wu, Xing Zhou, Wenbin Jiang, Hai Jin, and Xiangliang Zhang. Webcode2m: A real-world dataset for code generation from webpage designs. In Guodong Long, Michale Blumestein, Yi Chang, Liane Lewin-Eytan, Zi Helen Huang, and Elad Yom-Tov (eds.), *Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025*, pp. 1834–1845. ACM, 2025. doi: 10.1145/3696410.3714889. URL https://doi.org/10.1145/3696410.3714889.

Zehai He, Wenyi Hong, Zhen Yang, Ziyang Pan, Mingdao Liu, Xiaotao Gu, and Jie Tang. Vision2web: A hierarchical benchmark for visual website development with agent verification. *CoRR*, abs/2603.26648, 2026. doi: 10.48550/ARXIV.2603.26648. URL https://doi.org/10.48550/arXiv.2603.26648.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Qiming Li, Shujie Hu, Haohan Liu, Xiaocheng Feng, Songxiang Liu, and Guanglu Wan. Mtweb2code: Benchmarking coding agents on multi-turn regional reconstruction and localized modification. *CoRR*, abs/2608.03474, 2026. doi: 10.48550/ARXIV.2608.03474. URL https://doi.org/10.48550/arXiv.2608.03474.

Ryan Li, Yanzhe Zhang, and Diyi Yang. Sketch2code: Evaluating vision-language models for interactive web design prototyping. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 3921–3955. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.198. URL https://doi.org/10.18653/v1/2025.naacl-long.198.

Zhuoyao Liu, Zhengran Zeng, Shu-Dong Huang, Yang Liu, Shikun Zhang, and Wei Ye. GALA: multimodal graph alignment for bug localization in automated program repair. *CoRR*, abs/2604.08089, 2026. doi: 10.48550/ARXIV.2604.08089. URL https://doi.org/10.48550/arXiv.2604.08089.

Zimu Lu, Houxing Ren, Yunqiao Yang, Ke Wang, Zhuofan Zong, Junting Pan, Mingjie Zhan, and Hongsheng Li. Webgen-agent: Enhancing interactive website generation with multi-level feedback and step-level reinforcement learning. *CoRR*, abs/2509.22644, 2025. doi: 10.48550/ARXIV.2509.22644. URL https://doi.org/10.48550/arXiv.2509.22644.

OpenAI. Codex CLI. Software, 2025. URL https://github.com/openai/codex.

OpenAI. GPT-5.6 System Card, 2026. URL https://deploymentsafety.openai.com/gpt-5-6.

Qwen Team. Qwen3-vl technical report. *CoRR*, abs/2511.21631, 2025. doi: 10.48550/ARXIV.2511.21631. URL https://doi.org/10.48550/arXiv.2511.21631.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 3956–3974. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.NAACL-LONG.199. URL https://doi.org/10.18653/v1/2025.naacl-long.199.

Yuxuan Wan, Yi Dong, Jingyu Xiao, Yintong Huo, Wenxuan Wang, and Michael R. Lyu. Mrweb: An exploration of generating multi-page resource-aware web code from UI designs. *CoRR*, abs/2412.15310, 2024. doi: 10.48550/ARXIV.2412.15310. URL https://doi.org/10.48550/arXiv.2412.15310.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861. URL https://doi.org/10.1109/TIP.2003.819861.

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 multi-modal feedback. *CoRR*, abs/2601.04203, 2026. doi: 10.48550/ARXIV.2601.04203. URL https://doi.org/10.48550/arXiv.2601.04203.

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. *CoRR*, abs/2506.06251, 2025. doi: 10.48550/ARXIV.2506.06251. URL https://doi.org/10.48550/arXiv.2506.06251.

Jingyu Xiao, Zhongyi Zhang, Haoran Hou, Yuxuan Wan, Yuan Jiang, Yintong Huo, and Michael R. Lyu. Visualrepair: Dynamic tool calling and region focusing for visual software issue repair. *CoRR*, abs/2607.14075, 2026. doi: 10.48550/ARXIV.2607.14075. URL https://doi.org/10.48550/arXiv.2607.14075.

Qiao Xu, Yipeng Yu, Chengxiao Feng, and Xu Liu. 1d-bench: A benchmark for iterative UI code generation with visual feedback in real-world. *CoRR*, abs/2602.18548, 2026. doi: 10.48550/ARXIV.2602.18548. URL https://doi.org/10.48550/arXiv.2602.18548.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

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 Wang, and Ofir Press. Swe-bench multimodal: Do AI systems generalize to visual software domains? In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=riTiq3i21b.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 586–595. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00068. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_The_Unreasonable_Effectiveness_CVPR_2018_paper.html.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In Maria Christakis and Michael Pradel (eds.), *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, pp. 1592–1604. ACM, 2024. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

A Agent Instruction

The following instruction is used for all tasks and evaluated models, together with the task's feedback image. The agent operates in the corresponding project directory.

> Understand the editing intent communicated by the attached image, modify the code in the current project accordingly, and complete the task in a single turn. Inspect the project with read and search tools, then make the smallest necessary source edit. Do not run shell commands, install dependencies, start servers, or run builds, tests, linters, or formatters.

B Repository Resolution and Filtering

We collect 8,000 distinct candidate repositories per technology using language filters and framework keywords in the GitHub Repository Search API, yielding 32,000 candidate records. We verify technology assignments during repository resolution and filtering.

Table 3 summarizes Codex fallback usage and filtering outcomes, counting each candidate once per technology regardless of retries. Codex fallback was used for 6,395 candidate records, accounting for 20.0% of all records. Of these, 481 passed automated filtering. Subsequent project and task curation retained 21 of these repositories, contributing 99 benchmark tasks.

Table 3: Filtering outcomes and benchmark contributions of Codex-assisted candidates. "Filter pass" denotes passing automated filtering.

	Filtering		**Final benchmark**	
Technology	**Codex repos.**	**Filter pass**	**Final repos.**	**Tasks**
HTML	1,260	294	3	14
React	1,712	70	2	8
Vue	2,824	100	14	67
Angular	599	17	2	10
Total	6,395	481	21	99

C Task Distribution

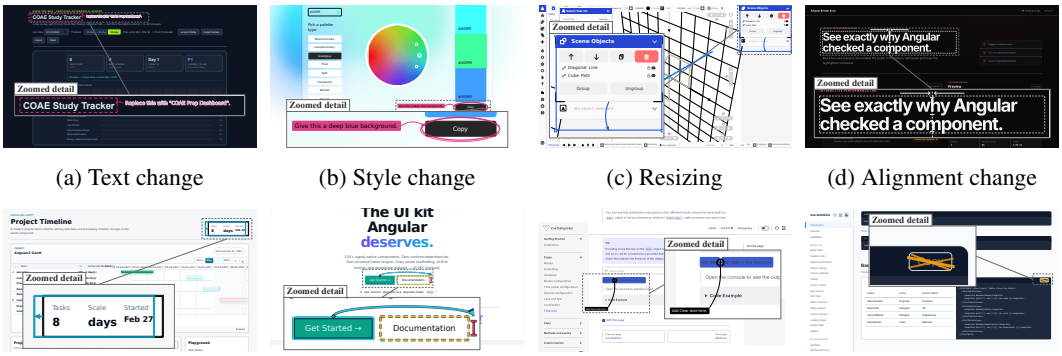
Table 4 reports task counts by edit operation and frontend technology. Each technology contributes 250 tasks covering all eight operations.

Table 4: Task distribution across edit operations and frontend technologies.

Operation	**HTML**	**React**	**Vue**	**Angular**	**Total**
Text change	28	28	27	29	112
Style change	26	24	27	26	103
Resizing	21	25	27	19	92
Alignment change	42	41	43	42	168
Component movement	29	35	32	34	130
Component reordering	33	30	28	31	122
Component addition	42	37	34	32	145
Component removal	29	30	32	37	128
Total	250	250	250	250	1,000

D Visual Feedback Examples

Figure 6 shows one example of visual feedback for each of the eight edit operations.



(a) Text change (b) Style change (c) Resizing (d) Alignment change

(e) Component movement (f) Component reordering (g) Component addition (h) Component removal

Figure 6: Visual feedback examples for the eight edit operations in ViRE-Bench. Each panel shows a feedback image with a magnified inset labeled "Zoomed detail." Gray boxes, connecting lines, and inset labels are added for illustration and are not part of the feedback provided to agents.

Sources. Panels (a–h) show interfaces rendered from coae-study-tracker, vue-color-wheel, dim, angular-render-scan, angular2-gantt, ngxsmk-ui-kit, vue-datepicker, and vue-slot-table, respectively. All eight repositories release their code under the MIT License. Feedback annotations were generated during task construction; magnified insets and gray guides were added for illustration.

E Edit-Intent Generation Prompts

When needed, the model first shortlists semantically suitable, non-redundant candidates using structured page and candidate data. It then selects a target using screenshots with highlighted candidates and accompanying metadata, returning parameters for the fixed edit operation according to a predefined schema.

Common selection instructions. The excerpt below presents the shared target-selection instructions, with `{operation}` replaced by the fixed edit operation. Task-specific inputs, including screenshots with highlighted candidates, viewport dimensions, and candidate metadata, are omitted. The full prompt also includes image-label instructions, operation-specific requirements, and annotation requirements where applicable.

> The edit operation is fixed: `{operation}`.
>
> Select the most semantically appropriate candidate that supports an objectively verifiable edit while preserving unrelated UI. Do not prefer a candidate merely because it is visually prominent, easy to locate in source code, or simple to implement. Do not favor candidate IDs or list order.

Operation-specific requirements. Table 5 summarizes the selection requirements and output parameters for all eight operations. Each response identifies a supplied candidate through `candidate_id` and specifies parameters subject to candidate-specific constraints where applicable. We validate and normalize these parameters before incorporating them into the VES. For text changes, style changes, and component additions, the model also returns a short textual annotation.

F Reference Implementation Details and VES Example

Source parsing and editing. We parse HTML with `parse5` and React JSX/TSX with `@babel/parser`. For Vue, we extract templates with `@vue/compiler-sfc` and parse them with `@vue/compiler-dom`. For Angular, we locate inline and external templates through component metadata in the Babel AST, then parse them with `@angular/compiler`.

We combine parser-reported source positions with temporary element identifiers to map rendered targets to source spans. Operation-specific editors modify these spans while preserving surrounding

Table 5: Selection requirements and output parameters for edit-intent generation. Parameter paths omit the enclosing `params` object.

Operation	Selection requirements and parameter output
Text change	Replace visible text with concise, page-appropriate content; return `new_text`.
Style change	Choose one conspicuous but coherent style change; return a style `property` and a `value` different from the current style.
Resizing	Choose a naturally resizable element and request a clearly visible size change; return `width_px` and `height_px`, using zero for at most one unchanged dimension.
Alignment change	Choose an alignment from the selected candidate's supplied allowed values; return `value`.
Component movement	Choose a self-contained component and a natural local adjustment; return `dx` and `dy` matching one supplied safe displacement.
Component reordering	Choose a component, a supplied sibling anchor, and its allowed ordering relation; return `anchor_candidate_id` and `relation`, without generating coordinates.
Component addition	Choose a supplied insertion slot and appropriate new content; return `slot_id` and `content`, with optional permitted `style_delta` fields.
Component removal	Choose a self-contained, dispensable element; return an empty parameter object.

source text. For shared-component edits, we modify the common source definition in React, Vue, and Angular projects. In plain HTML projects, we modify matched elements on each affected page. We remove temporary instrumentation before final rendering.

VES example. We illustrate the VES with a task from *angular2-gantt*[1]. The requested edit moves the upper-right statistics card 16 pixels to the left while preserving its size. Bounding boxes use the format $[x, y, \text{width}, \text{height}]$ in screenshot coordinates. We omit auxiliary metadata and repeated target records for brevity. The VES is withheld from agents in the standard benchmark setting.

[1]Source: `https://github.com/dalestone/angular2-gantt`, distributed under the MIT License.

```
{
  "context": {
    "viewport": {
      "width": 1365, "height": 768,
      "scroll_x": 0, "scroll_y": 0
    },
    "source": {
      "project_id": "project_1340",
      "framework": "angular", "route": "/"
    },
    "scope": "instance"
  },
  "edit": {
    "operation": "move_component",
    "target": {
      "node_id": "node_source_d41ffe663cdfcc",
      "bbox": [1105.875, 70, 216.625, 74]
    },
    "params": {
      "placement": {
        "type": "pixel_delta", "dx": -16, "dy": 0,
        "bbox": [1089.875, 70, 216.625, 74]
      }
    }
  },
  "feedback": {
    "geometry": {
      "result_bbox": [1089.875, 70, 216.625, 74]
    }
  }
}
```


Reference source edit. The reference editor applies relative positioning to the target element in `src/demo-app/demo-app.component.html`. The excerpt below abbreviates the unchanged class value and wraps the added line for readability. Agents are not required to use the same implementation as the reference.

```
- <div class="...">
+ <div style="position: relative; left: -16px; top: 0px"
+      class="...">
```

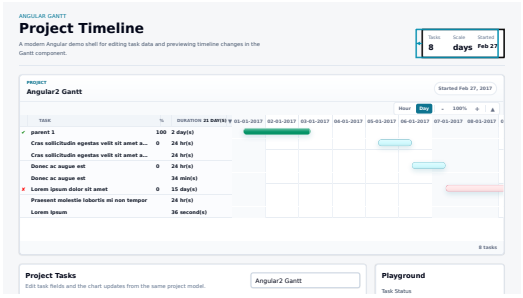
Figure 7 shows the corresponding visual feedback and reference result, with enlarged views of the edited region.

G Reference Stability and Region Partitioning

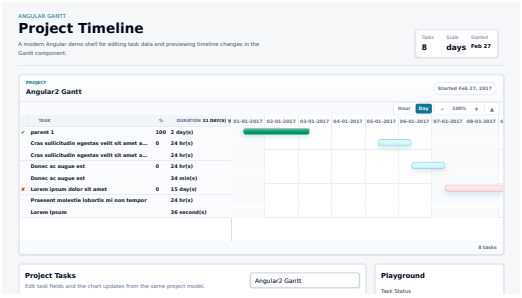
Figure 8 illustrates region partitioning for a task that removes the "Generate multiple" button from *Vue Color Avatar*[2].

We identify rendering instability using three captures each of the initial and reference projects. A pixel is marked as unstable if any RGB channel varies by more than 2 on the 0–255 scale across captures of either version. Stable differences between the initial and reference captures define the target region. We dilate both unstable and stable-difference regions by 3 pixels to accommodate the 7×7 SSIM window, with excluded regions taking precedence. Pixels outside the target and excluded regions form the preservation region.

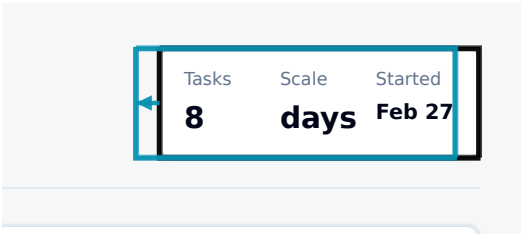
[2]Source: Vue Color Avatar (code: MIT License). Avatar artwork derives from *Avatar Illustration System* by Micah Lanier, licensed under CC BY 4.0. Screenshots are annotated and cropped for illustration.



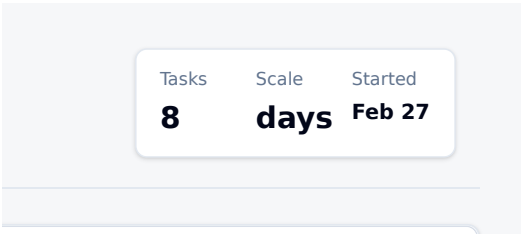
(a) Visual feedback



(b) Reference result

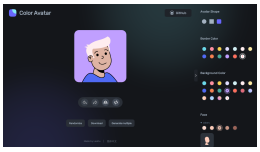


(c) Feedback detail

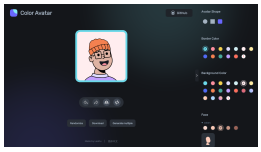


(d) Reference detail

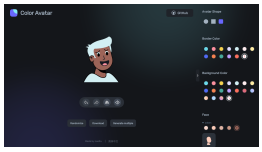
Figure 7: Visual feedback (a) and reference result (b) for moving the upper-right statistics card 16 pixels to the left while preserving its size. Panels (c) and (d) show the same region at the same magnification. The cyan outline marks the intended position, and the arrow indicates the movement direction.



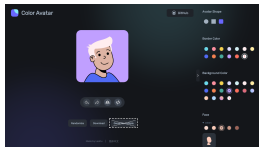
(a) Initial 1



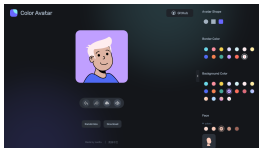
(b) Initial 2



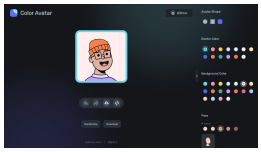
(c) Initial 3



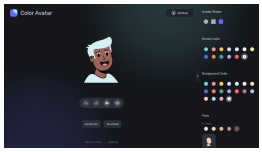
(d) Visual feedback



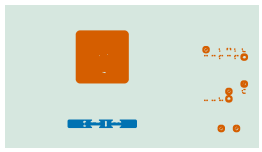
(e) Reference 1



(f) Reference 2



(g) Reference 3



(h) Region partition

Target Preservation Excluded

Figure 8: Reference stability and scoring regions. (a–c) Initial captures. (d) Visual feedback. (e–g) Reference captures. (h) Target (blue), preservation (pale green), and excluded (orange) regions. Repeated captures separate variable avatar content from the stable visual effect of button removal. Scoring masks are derived from the three initial and three reference captures, without using feedback markings or agent outputs.

H Operation-Specific Intent Checks

We evaluate intent satisfaction on the primary page using the editing specification and browser-extracted evidence from the initial, reference, and agent-modified projects. Target matching uses identity attributes, text, and structural context, excluding cues altered by the requested operation. Required elements must be matched unambiguously. Table 6 summarizes the standard conditions. Validity failure yields $J = 0$; source-level equivalence to the reference is not required.

Table 6: Standard intent checks on the primary page. All applicable conditions must hold; $J = 0$ if validity fails. Geometric thresholds are in pixels.

Operation	Passing condition
Text change	Target content matches the requested text after normalization.
Style change	Each requested property has similarity at least 0.75, accounting for visible effects where available.
Resizing	Each reference-changed dimension changes by at least 10 px in the same direction.
Alignment change	Text content, line count, and alignment anchors match the reference within a 1 px positional tolerance; computed alignment is used when geometry is not evaluable.
Component movement	The target center moves at least 10 px in the intended direction, with orthogonal drift $\leq$ 2 px and each size change $\leq \max(2\text{ px}, 5\%$ of the initial dimension$)$.
Component reordering	Target and anchor share a parent and satisfy the requested adjacency in DOM or reference-derived visual order.
Component addition	Requested content is added at the expected placement while existing matching elements remain.
Component removal	Required instances are removed without leaving visible target content or removing neighboring components preserved by the reference.

The evaluator accepts selected alternative implementations through operation-specific visual-equivalence or rendered-geometry checks. Where required, pixel equivalence means exact agreement with the reference within the stable primary-page target mask.

These checks assess operation satisfaction rather than exact visual reconstruction. Movement and resizing can pass even when displacement or final size differs from the reference; target fidelity measures the remaining visual discrepancy. Intent checks apply to the primary page, while target fidelity also covers other affected pages and preservation covers unrelated regions across evaluation pages.

I Human Validation

Protocol. Three master's students in computer science independently evaluate 96 tasks and 192 model submissions. The sample includes three tasks per operation–technology combination, two submissions per task, and 32 submissions per model. Model identities and automatic scores are hidden from the annotators. Categorical judgments require at least two matching definite responses. We aggregate overall-quality ratings using the median of at least two non-missing ratings. Responses indicating uncertainty or insufficient evidence are treated as missing.

Task quality. Annotators assess request clarity and contextual appropriateness using the initial screenshot and visual feedback. They then inspect the reference result for correctness and preservation of unrelated content. Each of these four criteria receives majority approval for 95 of 96 tasks and majority rejection for one task, with no unresolved judgments.

Intent satisfaction. Annotators judge whether each submission performs the requested operation on the correct target. Of the 177 submissions with $V = 1$, 173 receive definite human majority judgments, while four remain unresolved. Automatic intent checks agree with human majority

judgments on 166 of 173 submissions, yielding 96.0% agreement and Cohen's $\kappa = 0.918$. We exclude invalid submissions from this analysis to assess intent checking separately from validity gating.

Overall editing quality. Annotators rate overall quality on a 1–5 scale using primary-page renders, considering task completion and preservation of unrelated content. All 192 submissions have rendering evidence, including 15 with $V = 0$. Of these, 14 make no source changes and use the initial render; one fails a syntax check but remains renderable. Across all 192 submissions, automatic overall scores correlate with median human ratings, with Spearman's $\rho = 0.892$ and a 95% confidence interval of [0.847, 0.933]. We estimate the interval using 2,000 task-level bootstrap resamples.

J Validity Failures

Table 7 summarizes selected failed validity checks and budget-exhaustion status among invalid submissions across all six models. Budget exhaustion is recorded as a run status, not a validity check.

For DeepSeek, 227 of 231 invalid submissions contain no source-code changes, and 224 are marked as budget-exhausted. For Qwen, 99 of 148 invalid submissions contain no source-code changes, while 42 fail the build check. Thus, low validity can reflect the absence of source edits as well as failures in modified projects.

Table 7: Selected failed validity checks and budget-exhaustion status among invalid submissions. Entries are counts. Categories overlap and do not cover all validity checks; counts should not be summed.

Model	Invalid	No source change	Build failure	Budget exhausted
Gemini	22	21	1	18
GLM	33	29	3	27
GPT	8	4	4	0
DeepSeek	231	227	3	224
Claude	39	31	6	2
Qwen	148	99	42	3

K Visual Fidelity and Preservation by Edit Operation

Table 8 reports mean target fidelity and preservation by model and edit operation for submissions satisfying $V = J = 1$, alongside the qualifying counts. Because qualifying subsets differ across models, these conditional means describe visual quality within each subset and are not used for direct model ranking. Means based on very few qualifying submissions should be interpreted cautiously.

L Cross-Page Consistency in Shared-Component Tasks

For each shared-component submission satisfying $V = J = 1$, we compare primary-page target fidelity with the mean over other affected pages. We then average both quantities across qualifying submissions, giving each task equal weight. Scores are reported on a 0–100 scale.

Table 9 reports results for qualifying submissions from the 232 shared-component tasks. Qualifying subsets differ across models, so these comparisons are descriptive. The scores measure visual agreement with the reference across affected pages, not the proportion of tasks passing intent checks on every page.

M Diagnostic Experimental Details

Diagnostic subset. We first select 192 tasks independently of model outcomes. Within each operation–technology combination, we uniformly sample six distinct repositories and then uni-

Table 8: Mean target fidelity and preservation by edit operation, conditional on $V = J = 1$, with qualifying counts n. Scores are on a 0–100 scale; a dash indicates $n = 0$. Values rounded to 100.0 do not necessarily indicate exact agreement.

Operation	**Gemini**	**GLM**	**GPT**	**DeepSeek**	**Claude**	**Qwen**
Target fidelity (F)						
Text change	98.2	99.4	99.6	99.7	99.8	100.0
Style change	92.2	92.5	92.9	92.1	92.0	91.8
Resizing	75.5	76.1	83.8	74.2	66.7	–
Alignment change	99.6	99.6	99.1	98.7	100.0	–
Component movement	56.7	75.4	91.3	68.9	56.8	54.2
Component reordering	98.4	96.9	97.2	94.3	86.3	86.7
Component addition	91.1	87.4	88.9	87.3	86.7	92.4
Component removal	97.2	97.7	98.4	97.3	100.0	100.0
Preservation (P)						
Text change	99.5	99.8	99.9	100.0	100.0	99.9
Style change	100.0	100.0	100.0	100.0	99.9	99.8
Resizing	96.2	97.5	98.5	97.6	99.3	–
Alignment change	99.8	99.8	99.0	99.6	99.8	–
Component movement	96.5	96.6	97.4	95.2	97.9	88.7
Component reordering	99.7	99.8	99.9	100.0	98.6	100.0
Component addition	99.9	99.8	99.8	99.8	99.9	99.9
Component removal	99.4	99.0	99.4	99.3	99.9	100.0
Qualifying submissions (n)						
Text change	111	108	107	101	99	98
Style change	93	89	78	86	69	50
Resizing	28	30	25	7	7	0
Alignment change	118	83	63	43	33	0
Component movement	28	14	13	3	2	1
Component reordering	90	69	14	37	17	3
Component addition	117	110	92	89	66	48
Component removal	117	97	69	62	40	10

Table 9: Cross-page target fidelity, conditional on $V = J = 1$. Each qualifying task receives equal weight; n is the qualifying task count. "Other pages" denotes the within-task mean over other affected pages. Drop is Primary minus Other pages, computed before rounding.

Model	$\boldsymbol{n}$	**Primary**	**Other pages**	**Drop**
Gemini	156	95.14	89.83	5.31
GLM	131	95.99	92.75	3.24
GPT	90	95.96	91.64	4.32
DeepSeek	82	95.96	90.58	5.39
Claude	69	95.79	92.26	3.52
Qwen	35	98.09	93.91	4.18

formly select one task from each repository. We add 128 disjoint tasks, stratified by the number of models failing validity or intent checks under the standard input condition. The strata contain tasks with failures by 1–2, 3–4, or 5–6 of the six evaluated models. Selection also balances technologies and spreads tasks across repositories. The resulting subset contains 10 tasks per operation–technology combination, totaling 320 tasks, and remains fixed across all diagnostic models and conditions. We reuse standard input runs from the main evaluation and report one evaluated rollout per task for each supplemented condition.

Complete diagnostic results. Table 10 reports intent satisfaction, target fidelity, preservation, and overall score for all conditions. Target fidelity and preservation are averaged over submissions satisfying $V = J = 1$, whereas the mean task score includes all 320 tasks.

Table 10: Diagnostic results on the fixed 320-task subset. J is intent satisfaction in percent; F and P are conditional means over $V = J = 1$, with qualifying count n. S is the mean gated task score over all 320 tasks. F, P, and S are on a 0–100 scale.

Model	Condition	J	n	F	P	S
Gemini	Standard input	68.1	218	92.68	99.57	64.91
	+ Location	66.9	214	92.63	99.50	63.64
	+ Intent	95.0	304	97.78	99.72	93.60
	+ Intent + Location	97.8	313	98.39	99.78	96.76
GPT	Standard input	45.6	146	93.61	99.17	43.67
	+ Location	60.0	192	95.38	99.67	58.24
	+ Intent	95.3	305	98.45	99.80	94.33
	+ Intent + Location	96.6	309	98.63	99.90	95.71
Qwen	Standard input	16.9	54	97.12	99.66	16.55
	+ Location	23.4	75	94.88	99.99	22.72
	+ Intent	58.8	188	97.29	99.41	57.60
	+ Intent + Location	74.4	238	98.22	99.77	73.44

Source-location supplement. The **+ Location** supplement provides locations in the initial source code matched to the recorded target identities. Each entry contains a file path, a one-based line number, and a source excerpt of at most 300 characters. Shared components may map to a common source definition, whereas matched elements in separate HTML files may yield multiple locations. The supplement includes neither a reference patch nor implementation instructions.

Intent supplement. We construct the **+ Intent** supplement from the VES using fixed, operation-specific templates. It specifies the target's screenshot bounding box, verified visible text when available, and the requested operation and parameters. Relevant ordering or appearance references are also included. For resizing, it specifies the final visible outer-box dimensions measured from the reference render along the requested axes. The supplement instructs the agent to preserve unrelated content and appearance. It includes no source locations, reference patches, affected routes, or edit-scope labels.

Example. We use the movement task introduced in Appendix F, with visual feedback shown in Figure 7(a). The blocks below reproduce its source-location and intent supplements. Coordinates refer to the original 1365×768 screenshot. The combined condition appends the intent supplement followed by the source-location supplement.

+ Location supplement.

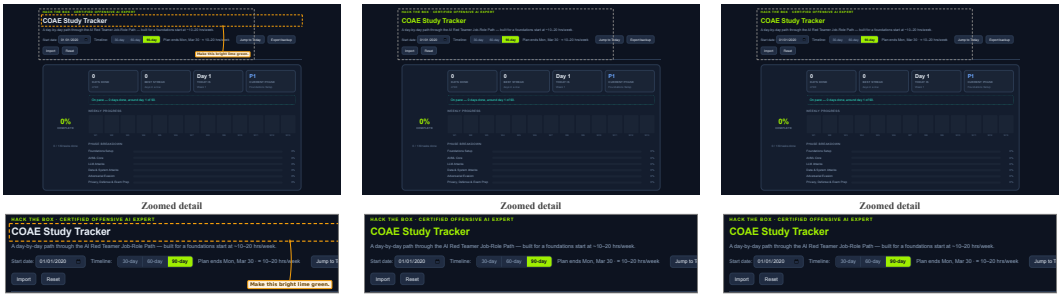
```
Source code related to the marked element(s) may be located here:

Line numbers are one-based in the original files.

File: src/demo-app/demo-app.component.html
Line: 12
Code:             <div class="grid grid-cols-3 gap-3 rounded-lg border
↪  border-slate-200 bg-white px-4 py-3 shadow-sm">
```

+ Intent supplement.

```
Additional information: intended visual edit.
Page: the page shown in the attached feedback image. Screenshot size:
↪  1365 x 768.
All element boxes below use coordinates in the supplied screenshot,
↪  not document coordinates.
Target: The element at screenshot box (x, y, width, height) =
↪  [1105.875, 70, 216.625, 74].
Requested change: Move the marked element by -16 pixels horizontally
↪  and 0 pixels vertically relative to its original position.
↪  Positive values mean right/down; negative values mean left/up.
↪  Preserve its content and size.
Preserve unrelated content and appearance; allow layout changes that
↪  naturally follow the requested edit.
```

N Qualitative Examples and Failure Analysis

Successful style editing. Figure 9 illustrates a successful style edit in *coae-study-tracker*.[3] The visual feedback highlights the "COAE Study Tracker" heading and requests a bright lime-green color. Under the standard input condition, Gemini-3.7-Flash adds `color: var(--green)` to the heading's CSS rule, reusing the existing color variable. The evaluated screenshot matches the reference pixel for pixel, with unrelated content and appearance preserved. The submission passes both validity and intent checks, with target fidelity, preservation, and overall scores of 100.



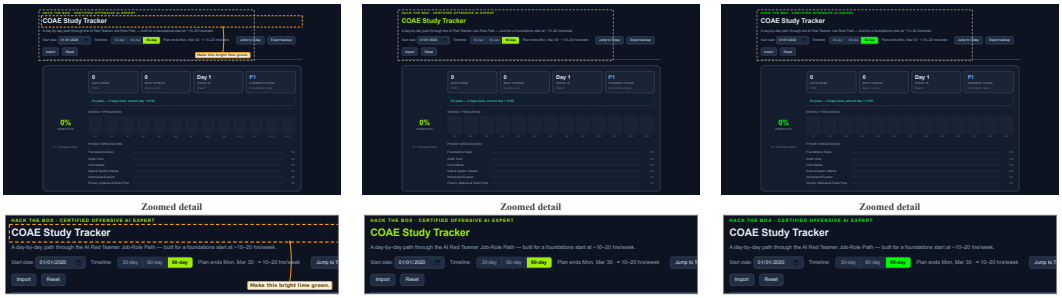
(a) Visual feedback (b) Reference result (c) Gemini-3.7-Flash

Figure 9: Successful style editing. The visual feedback (a) requests a bright lime-green heading. The reference (b) and model result (c) show the requested change with surrounding content preserved. Each panel includes a crop of the same region at the same magnification. Gray outlines and "Zoomed detail" labels are added for illustration.

Visual similarity without intent satisfaction. Figure 10 shows an unsuccessful attempt at the same heading-color task as Figure 9. Under the standard input condition, Qwen3-VL-235B-A22B-Thinking changes the global CSS variable `--green` from `#9fef00` to `#00ff00` without applying it to the heading. The heading retains its original color, while other elements using the variable

[3]Source: `https://github.com/mattrfield/coae-study-tracker`. The repository code is distributed under the MIT License.

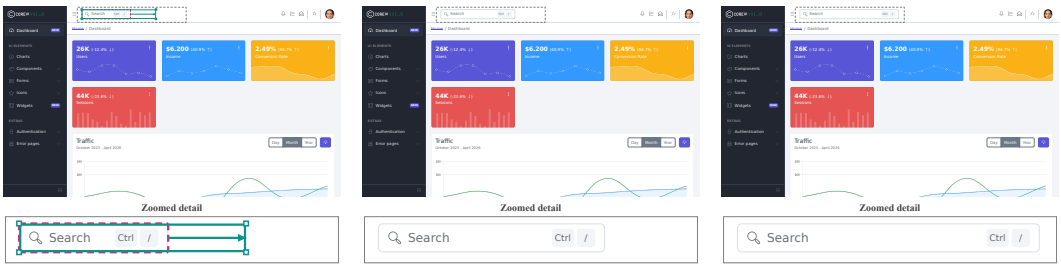
change color. The submission passes validity checks but fails the intent check. Target fidelity is 69.95 and preservation is 99.62, giving a harmonic mean of 82.19 before gating. Because the requested edit is not implemented, the intent gate sets the overall score to zero. This case illustrates the role of intent checks in distinguishing visual similarity from fulfillment of the requested edit.



(a) Visual feedback (b) Reference result (c) Qwen3-VL-235B-A22B-Thinking

Figure 10: Visual similarity without intent satisfaction. The feedback (a) requests a bright lime-green heading, as shown in the reference (b). The model result (c) leaves the heading color unchanged while changing other green elements. Each panel includes a crop of the same region at the same magnification. Gray outlines and "Zoomed detail" labels are added for illustration.

Correct resizing direction but inaccurate dimensions. Figure 11 illustrates inaccurate resizing in *coreui-free-vue-admin-template*.[4] The visual feedback requests widening the header search control from approximately 197 pixels to the marked width of 300 pixels. Under the standard input condition, GLM-5.3-Flash adds `min-width: 400px`, exceeding the reference width by 100 pixels. The submission passes validity checks and satisfies the intent check by increasing the width in the requested direction beyond the minimum threshold. Target fidelity is 75.90 and preservation is 99.65, yielding an overall score of 86.17. This case illustrates how target fidelity captures dimensional errors that remain after an edit passes the intent check.



(a) Visual feedback (b) Reference result (c) GLM-5.3-Flash

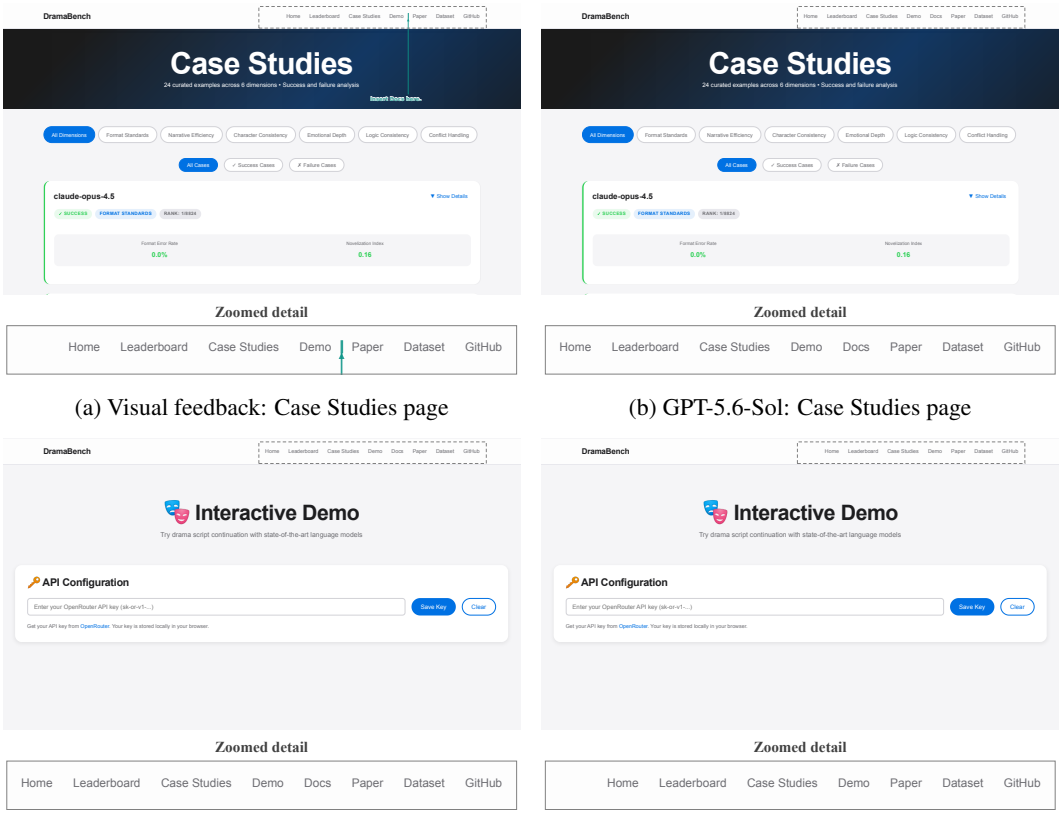
Figure 11: Inaccurate resizing despite intent satisfaction. The visual feedback (a) marks the requested search-control width, which the reference (b) matches but the model result (c) exceeds. Each panel includes a crop of the same region at the same magnification. Gray outlines and "Zoomed detail" labels are added for illustration.

Incomplete propagation across pages. Figure 12 illustrates incomplete propagation of a navigation edit in *DramaBench*.[5] The visual feedback requests adding a "Docs" link between "Demo" and "Paper" in the shared navigation bar. Under the standard input condition, GPT-5.6-Sol adds the link only to `web/cases.html`. The evaluated primary-page screenshot matches the reference pixel for pixel, whereas the home, demo, and leaderboard pages remain unchanged. The submission

[4]Source: `https://github.com/coreui/coreui-free-vue-admin-template`. The repository code is distributed under the MIT License.

[5]Source: `https://github.com/IIIIQIIII/DramaBench`. The repository code is distributed under the MIT License.

passes validity and primary-page intent checks, with target fidelity of 100 on the primary page but a mean of 42.96 across the other three affected pages. This case illustrates that a correct edit on the primary page does not guarantee consistent implementation across affected pages.



(a) Visual feedback: Case Studies page

(b) GPT-5.6-Sol: Case Studies page

(c) Reference: Demo page

(d) GPT-5.6-Sol: Demo page

Figure 12: Incomplete propagation across pages. The visual feedback (a) requests a "Docs" navigation link, which the model adds correctly on the primary page (b). On the Demo page, the link appears in the reference (c) but is absent from the model result (d). Each panel includes an enlarged navigation crop at the same magnification. Gray outlines and "Zoomed detail" labels are added for illustration.

Unintended changes outside the edit scope. Figure 13 illustrates unintended changes in *react808.com*.[6] The visual feedback requests left-aligning the page heading. Under the standard input condition, GPT-5.6-Sol deletes several global CSS rules from `src/index.css`, including layout and light-theme styles. The heading becomes left-aligned, but the page background and text colors also change, and the entire drum-machine interface shifts upward. Although the submission passes validity and intent checks, target fidelity is 44.72 and preservation is 43.79, yielding an overall score of 44.25. This case illustrates how preservation captures unintended changes outside the target region, even when the requested operation is satisfied.

Correcting an edit with explicit intent. Figure 14 compares the standard input and **+ Intent** conditions for a component-addition task in *angularx-qrcode*.[7] The visual feedback requests a new "Download SVG" button beside the existing "Download QR Code Image" button. Under the standard input condition, Gemini-3.7-Flash instead renames an existing link displayed only in SVG mode. The evaluated page remains unchanged in Canvas mode, and the submission passes validity

[6]Source: `https://github.com/PLUSDEVLLC/react808.com`. The repository code is distributed under the MIT License.

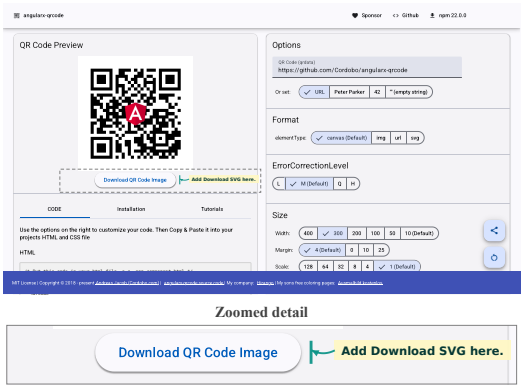
[7]Source: `https://github.com/Cordobo/angularx-qrcode`. The repository code is distributed under the MIT License.



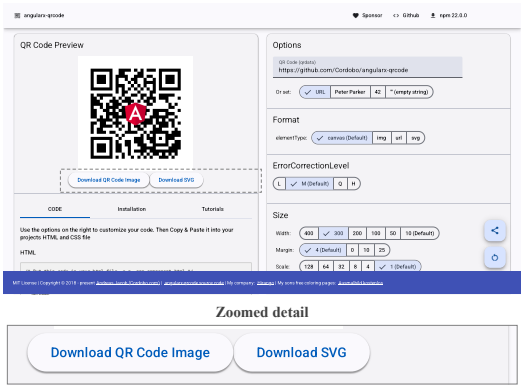
(a) Visual feedback (b) Reference result (c) GPT-5.6-Sol

Figure 13: Unintended changes outside the edit scope. The visual feedback (a) requests left alignment of the heading. The reference (b) preserves the surrounding interface, whereas the model result (c) also changes the background and text colors and shifts the interface upward. Each panel includes an enlarged header crop at the same magnification. Gray outlines and "Zoomed detail" labels are added for illustration.

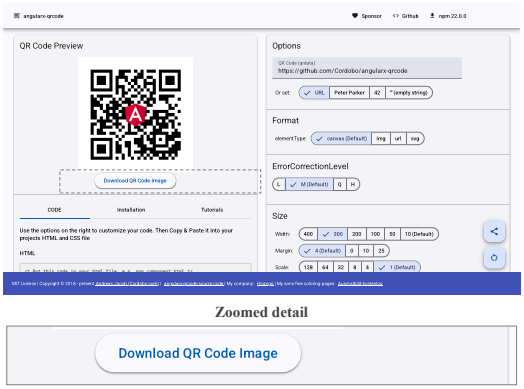
but fails the intent check. With **+ Intent**, the model receives an explicit specification of the new button, its insertion position, and its appearance reference. It then adds the requested button while retaining the existing control. Intent satisfaction changes from $J = 0$ to $J = 1$, and the overall score increases from 0 to 99.98. This case illustrates how explicit intent can help translate visual feedback into the requested edit.



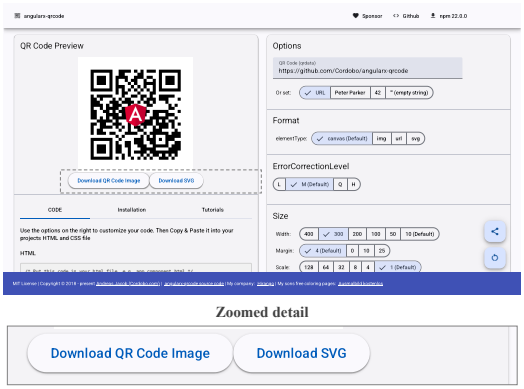
(a) Visual feedback



(b) Reference result



(c) Gemini-3.7-Flash: Standard input



(d) Gemini-3.7-Flash: + Intent

Figure 14: Correcting an edit with explicit intent. The visual feedback (a) requests a new "Download SVG" button beside the existing download control, as shown in the reference (b). The standard-input result (c) leaves the visible page unchanged, whereas the **+ Intent** result (d) includes the requested button. Each panel includes an enlarged view of the button area at the same magnification. Gray outlines and "Zoomed detail" labels are added for illustration.