WUIDGET: WEB UI GENERATION WITH SMALL MODELS VIA GAP-DRIVEN POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Web UI generation requires a model to produce executable code, coherent visual design, and working interactions in the same artifact. Small models fail differently across these axes, while aggregate benchmark scores provide little guidance about which training signal should address which failure. We propose *WUIDGET*, a gap-driven post-training framework that distills evaluation failures into a structured root-cause taxonomy and uses it to guide data and reward design across three stages. Supervised fine-tuning learns broad structural and aesthetic patterns; Rejection Fine-Tuning with Distillation (RFT-D) minimally repairs the model's own rollouts; and reinforcement learning optimizes verifier-grounded visual and interactive rewards. On OpenDesign (840 prompts), the final Qwen3-8B checkpoint improves over its base from 52.21 to 81.92 static ($+29.71$ points) and from 1.04 to 1.81 agentic (a 74% relative gain in working interactions per artifact), and improves over the comparably sized AesCoder-4B by $+4.18$ static. The static improvement over AesCoder remains consistent on WebDev Arena ($+3.05$) and Design Arena ($+3.33$). Across Qwen3-8B and SmolLM-3B, SFT supplies the largest visual-quality gain, RL the largest interactive gain, and inserting RFT-D changes the static–agentic trade-off reached by downstream RL. Matched gap/no-gap controls show that SFT guidance improves the macro-average agentic score by $+0.067$ (95% bootstrap interval $[+0.006, +0.128]$), while gap-selected RL queries produce a smaller, primarily static shift. These results support gap analysis not only as diagnosis, but as an organizing signal for specializing small models on multi-axis generation tasks.

1 INTRODUCTION

Generating a web UI from a natural-language request is more than code completion. The artifact must execute, exhibit a coherent visual hierarchy, and implement the requested interactions. These requirements fail independently: a page can render yet look generic, look polished yet omit requested behavior, or present plausible controls that do nothing. Most code benchmarks stress execution, while screenshot-to-code benchmarks emphasize fidelity to a reference design (Si et al., 2025; Yun et al., 2024; Li et al., 2025). Open-ended UI generation instead requires functionality, aesthetics, and interactivity in the same output, making it a useful test of post-training under multiple objectives.

The central difficulty is not a shortage of post-training methods, but deciding what each method should target. Supervised fine-tuning (SFT) can move the output distribution broadly, distillation can correct specific behavior, and reinforcement learning (RL) can optimize properties visible only after rendering. An aggregate benchmark score, however, does not say which residual failures dominate, what data should be collected next, or whether an intermediate stage changes what downstream RL can learn. Naively stacking methods therefore obscures both the reason for each intervention and the dependencies among them.

We ask whether evaluation failures can be converted into training signals that substantially improve models of at most 8B parameters. Compact specialists are practically relevant and scientifically revealing: their constraints expose which capability is acquired by which signal, and they can serve as role-specialized implementation agents within multi-agent systems (Wu et al., 2024). Our focus is the specialization recipe itself rather than a downstream product. We propose *WUIDGET*, a closed loop that groups evaluation critiques into a persistent root-cause taxonomy and uses that

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

taxonomy to guide successive interventions. SFT targets broad structural and aesthetic failures with teacher-authored demonstrations. Rejection Fine-Tuning with Distillation (RFT-D) then samples the student's own outputs and asks a teacher to accept, minimally repair, or reject each rollout, providing dense supervision for localized residual defects. RL finally optimizes rendered visual and interactive behavior. Re-evaluating each checkpoint exposes the next tier of failures and makes stage interactions measurable.

Under the same evaluation harness, the final Qwen3-8B specialist reaches the performance range of Claude Sonnet 4.5 across OpenDesign, WebDev Arena, and Design Arena. This is a contextual comparison, not a claim of uniform superiority: several differences are near evaluation noise and Sonnet is run without extended thinking. Against the comparably sized AesCoder-4B (Xiao et al., 2026), Qwen3-8B improves static quality by $+4.18$, $+3.05$, and $+3.33$ points across the three benchmarks. Across Qwen3-8B and SmolLM-3B, SFT repeatedly supplies the largest static gain, RL the largest agentic gain, and inserting RFT-D changes the static–agentic trade-off reached by downstream RL. Matched gap/no-gap controls further show that the taxonomy changes held-out behavior rather than merely describing it.

Our contributions are fourfold: (1) a repeatable framework that turns evaluation failures into stage-specific data and reward decisions; (2) RFT-D, a three-way teacher-repair stage anchored on student rollouts; (3) verifier-grounded visual and interactivity rewards designed for group-relative RL; and (4) stage-resolved evidence across two model families and three held-out benchmarks, including matched branches, gap/no-gap controls, a shared measurement ontology, and reward ablations.

2 Related Work

Web UI generation. Automatic interface generation has long treated layout as an optimization problem over users and devices (Gajos & Weld, 2004). Recent work instead learns to produce code from visual references: Design2Code (Si et al., 2025) and Web2Code (Yun et al., 2024) benchmark screenshot-to-HTML, while Sketch2Code (Li et al., 2025) turns rough sketches into prototypes. We generate from natural-language requests without a reference image, requiring open-ended aesthetic and behavioral quality. The closest work is AesCoder (Xiao et al., 2026), which couples web-code SFT with execution, aesthetic, and interactive rewards. We use its OpenDesign benchmark and compare with AesCoder-4B, but organize training around a failure taxonomy and resolve the contribution and interaction of three stages.

Learning from failures and distillation. Self-Instruct (Wang et al., 2023), STaR (Zelikman et al., 2022), Self-Refine (Madaan et al., 2023), and Reflexion (Shinn et al., 2023) reuse model-generated behavior or feedback. Self-Refine and Reflexion revise individual outputs at inference time; our gap analysis instead aggregates failures into a persistent checkpoint-level taxonomy that directs subsequent weight updates. Rejection-based self-training (Yuan et al., 2023; Gulcehre et al., 2023; Singh et al., 2024) retains successful sampled outputs, whereas distillation transfers teacher behavior (Hinton et al., 2015; Kim & Rush, 2016; Hsieh et al., 2023; Agarwal et al., 2024). Teacher-repair methods such as PERsD (Chen et al., 2023) and LEMA (An et al., 2024) are closest to RFT-D because they correct a student's failed output. RFT-D adds explicit accept and reject arms, constrains revision to localized repairs, and regenerates the reasoning trace so the corrected target reads as a coherent first attempt. This combines on-policy anchoring with a teacher's ability to inject behavior absent from sampled successes, while limiting the distribution shift of wholesale rewriting.

Model judges and compact specialization. LLM judges enable scalable evaluation and AI feedback (Zheng et al., 2023; Chiang et al., 2024; Bai et al., 2022; Lee et al., 2024; Gu et al., 2026), but exhibit position bias (Wang et al., 2024) and self-preference (Panickssery et al., 2024). WebDev-Judge (Li et al., 2026) studies critics for web development; we cite and execute its two query-filtering prompts, then filter according to their parsed outputs. Our RL design mitigates position bias and replaces saturated absolute visual scores with within-group comparisons. This matters specifically for group-relative RL: an accurate judge still supplies little learning signal when it assigns nearly identical scores to all sibling rollouts. More broadly, compact models can acquire strong narrow capabilities (Hsieh et al., 2023; DeepSeek-AI, 2025); our focus is how multiple signals jointly specialize them for functionality, aesthetics, and interactivity.

3 GAP-DRIVEN POST-TRAINING

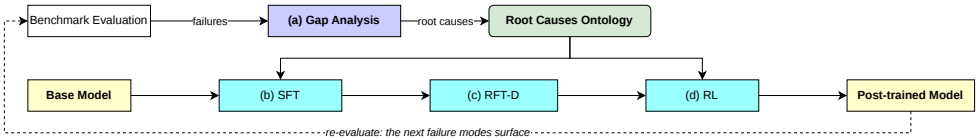


Figure 1: The WUIDGET loop. Evaluation failures form a root-cause ontology that guides SFT data and RL query and reward design; RFT-D repairs defects in individual rollouts. Every trained checkpoint returns to evaluation, exposing the next residual failures.

WUIDGET closes a loop between evaluation and training (Figure 1). A checkpoint is evaluated on held-out queries; written judge critiques are grouped into recurring root-cause categories; those categories guide the next data or reward intervention; and the resulting checkpoint is re-evaluated. Training taxonomies use validation splits. A separate 29-category measurement ontology is initialized from base test outputs, adds three categories first observed later, and is then applied retrospectively to all checkpoints; it never feeds training (§4).

Two resolutions of gaps. The taxonomy is a *population-level* instrument: it reveals which failures recur often enough to justify changing a data distribution or reward function. This is appropriate for SFT and RL, which commit to supervision before observing a particular output. RFT-D instead works at sample resolution. Once a rollout exists, a teacher can localize and repair its defect directly; routing an inert search bar through a population taxonomy adds no information. The two resolutions are complementary: the ontology has broad recall but cannot identify the defective span in one artifact, while the repair judge is precise per sample but blind to benchmark-level prevalence.

Why this order. Each stage uses the cheapest suitable signal. SFT first moves the model toward well-structured, visually designed interfaces through teacher-authored examples. RFT-D then anchors on the student's own distribution and provides dense supervision for localized defects, especially inert controls. RL runs last because artifact-level visual and behavioral quality cannot be demonstrated token by token and requires expensive rendered verification. The sequence thus alternates off-policy demonstrations, minimally repaired student rollouts, and fully on-policy optimization.

3.1 GAP ANALYSIS

For each checkpoint, the evaluation judge produces written critiques on a held-out query set. We consolidate recurring root causes into a named taxonomy and use their prevalence to allocate training attention. For SFT it selects public examples and drives category-specific synthesis; for RL it selects queries and motivates reward axes. The shared measurement ontology spans 29 coarse-to-fine categories. Appendix C.2 gives definitions and counts, and Appendix C.1 the trajectories. Section 5.3 compares matched arms with and without taxonomy-guided decisions.

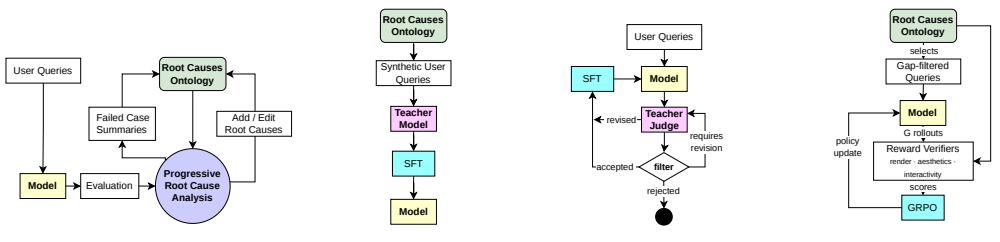


Figure 2: Mechanisms inside the loop: (a) root-cause analysis, (b) taxonomy-guided SFT synthesis, (c) RFT-D teacher triage into accept, revise, or reject, and (d) ontology-guided query selection plus verifier-grounded RL.

3.2 GAP-GUIDED SUPERVISED FINE-TUNING

SFT targets broad structural and aesthetic failures that teacher demonstrations convey more efficiently than on-policy search. Its roughly 110K-example mixture combines three sources. First, generic synthesized web UI and infographics examples are authored and filtered with DeepSeek-R1-0528 (DeepSeek-AI, 2025). Second, Kimi-K2.5 (Kimi Team, 2026) generates model-specific gap-focused examples category by category: for each named failure it proposes diverse queries likely to elicit that defect, writes complete HTML responses, and passes the pairs through the same evaluation judge; only examples above a quality threshold are retained. Third, 60K examples are selected from AesCode-358K (Xiao et al., 2026) according to the ontology rather than sampled uniformly. Each base model receives gap-focused data derived from its own failures.

The mixture is balanced so target-domain data is not swamped by the larger public corpus. Public examples also receive reasoning traces, matching the response format used at inference. We train for one epoch with an assistant-only masked causal objective using TRL (von Werra et al., 2020) and DeepSpeed ZeRO-3 (Rajbhandari et al., 2020); the exact mixture, objective, and masking are in Appendix A.1. The matched control in §5.3 keeps the example budget and optimization fixed while replacing ontology-selected and gap-focused blocks with unguided data.

Masking the prompt is important because the three sources use visibly different prompt styles: otherwise part of the objective would model artifacts of data construction rather than tokens the model must generate. The stage is therefore conventional in optimization but targeted in distribution; the control tests that distinction while holding training volume fixed.

3.3 REJECTION FINE-TUNING WITH DISTILLATION

SFT improves surface form but often leaves controls visually plausible and behaviorally inert. RFT-D targets these localized defects while remaining close to the student's distribution. Given a WebSight design brief (Laurençon et al., 2024), the current SFT checkpoint samples $N = 8$ rollouts for Qwen3-8B and $N = 4$ for SmolLM-3B. Kimi-K2.5 then assigns each rollout to one of three arms:

1. **Accept:** retain an output that already meets the quality bar.
2. **Revise:** repair identifiable defects under a minimal-edit instruction that preserves layout, styling, and unaffected code.
3. **Reject:** discard outputs that are poor or cannot be fixed within that constraint.

For revised samples, the teacher regenerates a fresh first-person reasoning trace leading to the corrected code. Otherwise the trace would describe the defective implementation or narrate a revision process unavailable at inference. The repair prompt requires failure status when bounded repair is impossible and forbids revision-revealing language; prompts and checks appear in Appendix E. Accepted and revised rollouts form $\mathcal{D}_{\mathrm{RFT-D}}$ and are trained for one epoch with the same response-masked causal objective as SFT (Appendix A.2).

Why minimal repair matters. Minimal editing is a design principle rather than a measured localization guarantee. It is operationalized by the repair prompt, which requires failure status when a defect cannot be fixed while preserving the original layout, styling, and unaffected code; such samples enter the reject arm rather than being rewritten from scratch. Because most retained tokens originated from the student, they should already have relatively high likelihood, making teacher-introduced behavior disproportionately informative while limiting wholesale imitation of teacher style. The accept/revise/reject routing also distinguishes RFT-D from pure rejection sampling, which cannot inject absent behavior, and unconstrained teacher rewriting, which can revert to ordinary off-policy distillation. RFT-D sits between SFT and RL because event-handler wiring admits direct correction but is expensive to discover through artifact-level exploration. Its immediate score gain is modest; its more important empirical role is changing the static–agentic endpoint reached by downstream RL (§5.2).

3.4 VERIFIER-GROUNDED REINFORCEMENT LEARNING

RFT-D still learns from target sequences rather than measurements of finished artifacts. The remaining failures are properties that no single target token expresses: relative visual quality among plausible pages and whether requested controls produce observable behavior. We optimize these with GRPO (Shao et al., 2024) in verl (Sheng et al., 2025), sampling groups of $G = 8$ rollouts and standardizing each rollout's reward within its group. Each update uses one newly sampled group once, so the signal is fully on-policy and every token in a rollout receives the same artifact-level advantage. Qwen3-8B uses two pointwise-reward epochs followed by six pairwise-reward epochs; SmolLM-3B uses one plus three. The curriculum changes only the visual term at the phase boundary, leaving render and interactivity fixed. We use no KL penalty or entropy bonus; the objective and query accounting are in Appendix A.3.

Queries and reward suite. Training queries come from chronologically earlier Design Arena traffic. Gap selection retains queries representing validation-set failure categories; safety and intention prompts from WebDevJudge (Li et al., 2026) are then executed, and their parsed outputs remove `UNSAFE` or intention-below-3 queries. These are the two prompts we cite; filtering is our downstream decision rule over their outputs. The exact prompts and parser are reproduced in Appendix E.1. Filtering leaves 1,368 training and 337 validation queries. The independently filtered test split retains 511 prompts and receives no gap selection.

Each rollout receives render, visual, and interactivity rewards,

$$r = \mathrm{clip}_{[0,1]}(0.2r_{\mathrm{render}} + 0.4r_{\mathrm{vis}} + 0.4r_{\mathrm{ix}})\,. \tag{1}$$

In the pointwise phase, $r_{\mathrm{vis}} = \mathrm{clip}_{[-3,10]}(s)/10$ for judge score s; the resulting -0.3 fallback penalty motivates the outer clip. The render term uses Playwright errors, warnings, and element count, but saturates once most rollouts emit valid HTML. Late-stage discrimination therefore comes mainly from the judged terms. In group-relative learning, useful reward depends on *within-group spread*, not absolute calibration: bunched sibling scores provide little gradient. Pairwise visual ranking and task-conditioned, closed-scale interactivity grading widen this spread.

Pairwise visual reward. The first RL phase uses a pointwise screenshot judge. In the second phase we replace only that term with pairwise comparison because absolute scores saturate as quality rises. The judge compares screenshots within a rollout group, the outcomes induce dense preference ranks, and ranks are mapped so the best tier receives 1, the worst 0, and ties share a score:

$$s_i = \frac{r_{\max} - r_i}{r_{\max} - 1} \quad (r_{\max} > 1), \qquad s_i = 0.5 \text{ otherwise.} \tag{2}$$

A comparator only needs to choose which artifact is better; it need not maintain a stable numeric scale throughout training. To mitigate position bias (Wang et al., 2024), each pair is judged in an A-B-B-A schedule: every rollout appears twice in each slot, at least three votes decide a winner, and a 2–2 split is a tie. We precompute all $\binom{8}{2}$ pairs because independent calls saturate available concurrency; call and serial-wave accounting is in Appendix D.1.

Two-turn interactivity reward. A live browser agent is the most faithful check of interaction, but at roughly 135 seconds per rollout it is impractical inside the training loop. We replace it with a 10–15 second static judge over screenshot and source. A single turn would entangle two questions grounded in different evidence: what the requested page *should* make interactive, and whether the implementation supplies it. It would also let an under-built page define its own grading target by exposing only the interactions it happened to implement.

We therefore separate expectation from verification. Turn 1 sees the prompt and screenshot, but not the source, and assigns an expected-interactivity tier (`none`, `light`, or `rich`) plus the components and behaviors that should exist. Turn 2 receives the same evidence, Turn 1's output verbatim, and the HTML, then grades each named component on a closed enum: `working` (1.0), `partial` (0.6), `broken` (0.2), or `missing` (0). Scores use a tier-specific capacity rather than the number of named components, preventing full credit for under-delivery. Appendix D.3 gives the score and example; Appendix E.2 gives the prompts. Section 5.3 compares the live agent.

4 EVALUATION SETUP

Models and benchmarks. We post-train Qwen3-8B and SmolLM-3B with model-specific gap data and budgets; their agreement tests recurrence, not controlled scaling. DeepSeek-R1-0528 authors generic synthesized SFT examples, while Kimi-K2.5 performs model-specific gap analysis, synthesis, and RFT-D repair. We evaluate on three held-out benchmarks from distinct sources so that the result does not rest on one prompt provenance. *OpenDesign* (Xiao et al., 2026) contains 840 prompts across general websites, 3D design, games, UI components, and data visualization. *WebDev Arena* contributes 500 prompts sampled from its $10{,}501$-battle public preference dataset (LMArena, 2025), preserving the category mix of Design Arena traffic. *Design Arena* contributes 511 of its $1{,}000$ most recent public conversations after the safety and intention filtering in §3.4; unlike the two test-only sets, its chronologically earlier traffic also supplies RL queries.

References and provenance. Claude Sonnet 4.5 is the main performance-range reference: snapshot `claude-sonnet-4-5-20250929` is evaluated with the same harness and decoding settings, but with extended thinking disabled while Qwen uses its default thinking mode. Its row contextualizes the achieved range rather than providing a controlled reasoning-budget comparison. AesCoder-4B (Xiao et al., 2026) is the closest comparably sized prior-art baseline. The same work supplies AesCoder, OpenDesign, the AesCode corpus from which our SFT mixture draws 60K examples, and the released evaluation harness. We therefore enforce prompt disjointness between selected AesCode examples and OpenDesign and score every reported model identically; the comparison is conservative in the useful sense that it is measured on the baseline authors' benchmark.

Splits and leakage. OpenDesign and WebDev Arena are test-only. Design Arena is partitioned chronologically, reserving the most recent requests for test. Training taxonomies use held-out validation failures: synthesized-data validation guides SFT, and Design Arena validation guides RL query selection. A separate 29-category measurement ontology is initialized from base OpenDesign outputs, adds three later-only categories, and is then applied retrospectively to all checkpoints; it never selects training data. The synthesized corpus is independently split, and selected AesCode prompts are disjoint from OpenDesign.

Scoring and protocol. We retain OpenDesign's two axes. The **static score** (0–100) is an LLM-as-judge assessment of the rendered screenshot for prompt alignment, aesthetic quality, and structure. The **agentic score** is the average count of successful interactions found by a browser agent; it rewards both correctness and breadth, while pages with no interactive elements contribute zero. Every cell uses Claude Opus 4.8 as judge and interaction agent under one generation protocol. We preserve the published rubrics but separate browser crashes and timeouts from genuine interaction failures, which the released harness records alike as zeros. These failures receive no score rather than entering the mean as genuine zeros. Absolute scores are consequently not comparable with values produced by the original default configuration, but all rows here remain directly comparable. Static per-sample standard deviation is approximately 19, implying a 0.65–0.85 standard error for these benchmark sizes; repeated agentic scoring moved the mean by roughly 0.05. Inference settings are in Appendix D.4; harness changes and scoring diagnostics are in Appendix D.5.

5 RESULTS

5.1 OVERALL PERFORMANCE

Table 1 traces both post-training paths and anchors the external comparisons. Under the same harness, final Qwen3-8B differs from Claude Sonnet 4.5 by $+1.04/-0.02/+0.58$ static and $+0.11/0.00/+0.24$ agentic across the three benchmarks. This supports performance-range, not uniform-superiority, language because two static differences are sub-point and reasoning budgets differ. Against comparable-scale AesCoder-4B, final Qwen improves static quality consistently by $+4.18/+3.05/+3.33$; agentic differences are small, and direct SFT $\rightarrow$ RL is the strongest Qwen checkpoint on that axis throughout. Both model panels improve sharply and share the broad stage ordering, but model-specific data and budgets preclude a controlled scaling claim.

Table 1: Design-generation results on held-out OpenDesign (840 prompts), WebDev Arena (500), and Design Arena (511). *Static* is LLM-as-Judge visual quality (0–100); *Agentic* is OpenDesign's per-interaction success count averaged over successfully scored samples. Panels (a) and (b) trace model-specific post-training paths. Within each model panel, the best stage is bold and the second-best is underlined; reference rows are unranked. SFT → RL branches from SFT; for the other post-training rows, ▲ / ▼ show the change from the preceding checkpoint. Evaluation settings and test splits are shared (§4).

	OpenDesign		WebDev Arena		Design Arena	
Model	Static	Agentic	Static	Agentic	Static	Agentic
(a) Qwen3-8B						
Base Qwen3-8B	52.21	1.04	60.27	1.20	58.71	1.02
SFT	73.85 ▲21.64	1.15 ▲0.11	76.62 ▲16.35	1.14 ▼0.06	73.81 ▲15.10	1.14 ▲0.12
SFT → RFT-D	75.22 ▲1.37	1.34 ▲0.19	77.52 ▲0.90	1.37 ▲0.23	76.28 ▲2.47	1.35 ▲0.21
SFT → RL *(branch)*	<u>76.68</u>	**1.86**	<u>78.99</u>	**1.84**	<u>77.53</u>	**1.81**
SFT → RFT-D → RL *(final)*	**81.92** ▲6.70	<u>1.81</u> ▲0.47	**84.66** ▲7.14	<u>1.75</u> ▲0.38	**82.49** ▲6.21	<u>1.74</u> ▲0.39
(b) SmolLM-3B						
Base SmolLM-3B	23.87	0.26	26.90	0.32	26.98	0.22
SFT	61.27 ▲37.40	0.76 ▲0.50	62.31 ▲35.41	0.75 ▲0.43	60.29 ▲33.31	0.69 ▲0.47
SFT → RFT-D	62.78 ▲1.51	0.89 ▲0.13	63.98 ▲1.67	0.88 ▲0.13	62.26 ▲1.97	0.98 ▲0.29
SFT → RL *(branch)*	**81.95**	<u>1.44</u>	**80.09**	<u>1.36</u>	**79.41**	<u>1.43</u>
SFT → RFT-D → RL *(final)*	<u>76.78</u> ▲14.00	**1.73** ▲0.84	<u>75.40</u> ▲11.42	**1.65** ▲0.77	<u>74.31</u> ▲12.05	**1.77** ▲0.79
References						
Claude Sonnet 4.5	80.88	1.70	84.68	1.75	81.91	1.50
AesCoder-4B	77.74	1.73	81.61	1.79	79.16	1.79

5.2 Stage Contributions

Across both models and all benchmarks, SFT supplies the largest sequential static gain, RFT-D the smallest, and RL the largest agentic gain. For Qwen3-8B, SFT contributes $+15.10$–$+21.64$ static points, whereas RL after RFT-D contributes $+0.38$–$+0.47$ agentic; SmolLM-3B follows the same ordering at a different scale. RFT-D nevertheless changes downstream RL's endpoint. Relative to direct RL, it gives Qwen roughly five additional static points with a 0.05–0.09 agentic trade-off, while giving SmolLM roughly 0.3 additional agentic points with a 4.69–5.17 static trade-off. These opposite directions rule out a universal gain on either metric but establish initialization-dependent rebalancing.

For Qwen3-8B, RFT-D's immediate OpenDesign static gain is only $+1.37$, but a paired comparison over the 839 samples scored at both stages gives a 95% interval of $[+0.07, +2.63]$, and outputs missing a closing `</html>` fall from 17 to 5. The stage is therefore not inert even before RL, but its larger effect remains path dependence rather than adjacent-score improvement.

RFT-D also adds 0.19–0.23 agentic at its own Qwen checkpoint, before RL adds another 0.38–0.47. This is consistent with dense repair as a useful intermediate signal rather than a sufficient replacement for verifier-grounded optimization.

The common measurement ontology corroborates the score-level account. Gap instances fall monotonically by 32.6% and clean generations rise from 0 to 149 of 840 (Table 2). SFT removes many coarse structural and aesthetic failures but does not solve missing dynamics; later stages reduce the aggregate further, and RL supplies the largest agentic gain. Some fine-grained categories increase as outputs become more elaborate, so monotonic aggregate closure does not imply every named failure improves at every stage. The full category trajectories are reported in Appendix C.1.

The scoring diagnostics in §4 also limit the weight placed on close rows: sub-point static differences among strong checkpoints do not support directional claims on one benchmark, and agentic differences below roughly 0.05 are within observed re-scoring variation.

Table 2: Failure evolution on OpenDesign under the common 29-category measurement ontology. Categories may overlap; *clean* means no identified gap. Appendix C.2 gives all categories.

	Base	SFT	RFT-D	RL
Gap instances	1,463	1,149	1,084	986
Gaps per generation	1.74	1.37	1.29	1.17
Missing requested dynamics	37	50	56	49
Clean generations (of 840)	0	93	116	149

5.3 Controlled Ablations

Gap/no-gap controls. Two matched controls test whether the taxonomy changes learned behavior: one removes gap guidance from SFT data construction, and the other replaces ontology-based RL query selection with uniform sampling. All arms share held-out prompts; intervals come from a stratified prompt bootstrap. Table 3 reports gap-guided minus no-gap differences. Data-overlap calculations and linear-response sensitivity estimates are deferred to Appendix B.2.

For SFT, both arms use the same base, 110K-example budget, epoch count, and optimization; the no-gap arm replaces ontology-selected and gap-focused blocks with unguided AesCode examples, jointly ablating selection and synthesis. Guidance improves agentic score on every benchmark, with macro $+0.067$ $[+0.006, +0.128]$, while the static interval includes zero. For RL, both arms start from the same RFT-D checkpoint and differ only in ontology-based query selection; both retain the gap-informed reward, so this control does not ablate the complete gap-driven RL design. The macro static shift is $+0.63$ $[-0.02, +1.27]$ and the agentic shift is $+0.036$ $[-0.032, +0.104]$. The effect is smaller and primarily static, with no reliable uniform gain. Each comparison has one training run per arm, so intervals quantify prompt and evaluation uncertainty, not seed variance.

Together, the controls support the framework's central operational claim rather than a generic benefit from the word "gap": the same ontology changes different stages through different mechanisms and produces different held-out effects. They do not establish that every gap-guided decision helps, or that the observed magnitudes survive a change of training seed.

Table 3: Matched gap/no-gap controls. Entries are absolute differences (gap-guided − no-gap), so positive values favor guidance; Static and Agentic remain in native, non-comparable units. Macro averages weight benchmarks equally; brackets are 95% stratified prompt-bootstrap intervals. Bold marks the only macro effect whose interval excludes zero.

Stage	Axis	OpenD.	WebDev	Design	Macro	95% interval
SFT	Static	-0.39	$+0.56$	$+1.32$	$+0.49$	$[-0.57, +1.55]$
	Agentic	$+0.093$	$+0.059$	$+0.049$	$+0.067$	$[\mathbf{+0.006}, \mathbf{+0.128}]$
RL	Static	-0.20	$+1.39$	$+0.70$	$+0.63$	$[-0.02, +1.27]$
	Agentic	$+0.090$	-0.016	$+0.034$	$+0.036$	$[-0.032, +0.104]$

Reward design controls. Table 4 isolates two reward choices using branches from the same SFT checkpoint. Pairwise ranking adds 1.08 static points over pointwise scoring, with paired 95% interval $[+0.19, +1.97]$. It costs 112 judge calls per eight-rollout group versus eight pointwise calls, but independent comparisons finish in seven serial waves under our concurrency cap (Appendix D.1). Visual-only RL improves static quality while reducing agentic score relative to SFT, showing the cost of omitting interactivity from the objective.

The interactivity ablation holds render and pointwise visual terms fixed. Replacing the live browser agent with the two-turn judge costs 3.89 static points; the judge-minus-live agentic difference is -0.08 with interval $[-0.18, +0.02]$, so it does not establish a difference in elicited interactivity. The substitution buys feasibility: approximately 10–15 seconds rather than 135 seconds per rollout. We therefore treat the judge as a deliberate throughput–quality compromise, not a free substitute. These variants were selected at the SFT checkpoint and carried onto RFT-D without re-running the ablation, so their ordering is established only at SFT. Additional paired diagnostics are in Appendix B.3.

Table 4: Reward-recipe ablations on OpenDesign. Each pair starts from the same SFT checkpoint and changes one reward term; bold marks the better value on the targeted metric. †Neither visual-only arm optimizes interactivity, so both regress from SFT on Agentic. ‡The interactivity arms share the same visual reward; the judged arm's lower Static score is therefore an off-axis regression.

Reward composition	Static	Agentic
SFT checkpoint (shared starting point)	73.85	1.15
Visual reward: pointwise vs. pairwise		
render + pointwise visual	80.70	0.91†
render + pairwise visual	**81.78**	0.99†
Interactivity signal: live agent vs. judge		
render + pointwise visual + agentic ix	80.54	**2.03**
render + pointwise visual + judged ix	76.65‡	1.95

6 Discussion and Limitations

RFT-D's clearest effect is the trajectory it gives subsequent RL, consistent with evidence that RL post-training depends strongly on initialization (Chu et al., 2025; Kang et al., 2026). The direction is model-dependent: the RFT-D-initialized Qwen path reaches substantially higher static quality with a small agentic trade-off, whereas the SmolLM path reaches higher agentic scores with a larger static trade-off. Minimal-edit supervision may shift probability toward targeted repairs before RL applies coarse sequence-level advantages, placing the same reward in a different local policy region. The branch comparison establishes this rebalancing, not the proposed mechanism or a universal direction. The small adjacent RFT-D delta makes the stage look dispensable; the downstream branches instead show that it changes which endpoint RL reaches.

This interpretation is compatible with feature-level evidence that RL can preserve more of a base model's representation than SFT while continuing to evolve during post-training (Shi et al., 2026). Whether RFT-D creates an analogous preconditioning shift here is untested; resolving that mechanism would require representation-level or token-level intervention rather than another endpoint comparison.

Five limitations bound the interpretation. The model families use different gap data, rollout counts, and curricula, so agreement demonstrates recurrence rather than controlled scaling. Reward variants were selected at SFT and not repeated after RFT-D. The judge-derived ontology may redirect attention as coarse failures disappear, although total gaps fall and clean generations rise under the common measurement taxonomy. Each control has one run per arm, so intervals exclude training-seed variance; the SFT control also jointly removes subset selection and targeted synthesis. Sonnet comparisons are contextual because some differences approach evaluation noise and inference reasoning settings differ. The overlap-normalized control estimates additionally assume linear response to the fraction of changed data and are sensitivity analyses, not measured counterfactuals. We also do not measure serving cost or latency, conduct user studies, or evaluate the specialist inside a complete multi-agent system; those benefits remain future work.

7 Conclusion

WUIDGET turns recurring evaluation failures into stage-specific data and reward decisions. Across two compact model families, SFT contributes most visual improvement, RL most interactive improvement, and RFT-D changes the downstream trade-off. The final Qwen3-8B specialist reaches Claude Sonnet 4.5's performance range under a shared harness and consistently exceeds the comparably sized AesCoder-4B on static quality, while matched controls show that gap guidance itself changes held-out behavior. More broadly, the results support evaluation gaps as an organizing signal for building specialist models when quality is multi-axis and the appropriate supervision changes over the course of post-training. The framework is therefore a way to decide not only what a specialist still lacks, but which form of supervision is best suited to the remaining failure.

AI USE STATEMENT

Generative AI was used to implement, evaluate, and document human-designed methods. WUIDGET, RFT-D, the reward functions, hypotheses, and experiments were conceived by the authors. As detailed in the paper, DeepSeek-R1-0528 generated and filtered generic SFT data; Kimi-K2.5 performed model-specific gap analysis, generated gap-focused data, and repaired rollouts for RFT-D; and Claude Opus 4.8 served as the evaluation judge and interaction agent. Claude Code assisted with experimental code, data processing, figures, and polishing author-written text without changing its substantive content. Codex supported a second revision pass, including reference auditing, rephrasing, restructuring, and compression. AI also clarified analyses first produced by the authors, but did not independently derive interpretations from raw results. All AI-assisted outputs were manually reviewed, with code testing, citation checking, and numerical validation performed by the authors. AI was not used to formulate mathematical claims or proofs, translate text, conduct surveys or interviews, or perform transcription. The authors take full responsibility for the work and its contents.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5be69a584901a26c521c2b51e40a4c20-Paper-Conference.pdf.

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. Learning from mistakes makes LLM better reasoner, 2024. URL https://arxiv.org/abs/2310.20689.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional AI: Harmlessness from AI feedback, 2022. URL https://arxiv.org/abs/2212.08073.

Hailin Chen, Amrita Saha, Steven Hoi, and Shafiq Joty. Personalized distillation: Empowering open-sourced LLMs with adaptive learning for code generation. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6737–6749, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.417. URL https://aclanthology.org/2023.emnlp-main.417/.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating LLMs by human preference. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 8359–8388. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/chiang24b.html.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 10818–10838. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/chu25c.html.

DeepSeek-AI. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

Krzysztof Gajos and Daniel S. Weld. SUPPLE: automatically generating user interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, IUI '04, pp. 93–100, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138156. doi: 10.1145/964442.964461. URL https://doi.org/10.1145/964442.964461.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Zhouchi Lin, Bowen Zhang, Lionel Ni, Wen Gao, Yuanzhuo Wang, and Jian Guo. A survey on LLM-as-a-judge. *The Innovation*, 7(6):101253, June 2026. ISSN 2666-6758. doi: 10.1016/j.xinn.2025.101253. URL https://doi.org/10.1016/j.xinn.2025.101253.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling, 2023. URL https://arxiv.org/abs/2308.08998.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507/.

Feiyang Kang, Michael Kuchnik, Karthik Padthe, Marin Vlastelica, Ruoxi Jia, Carole-Jean Wu, and Newsha Ardalani. Quagmires in SFT-RL post-training: When high SFT scores mislead and what to use instead. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 56876–56918, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/5d6ae8ba43ecb378030753c4408ef9bd-Paper-Conference.pdf.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1317–1327, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1139. URL https://aclanthology.org/D16-1139/.

Kimi Team. Kimi K2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of Web screenshots into HTML code with the WebSight dataset, 2024. URL https://arxiv.org/abs/2403.09029.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Ren Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. RLAIF vs. RLHF: Scaling reinforcement learning from human feedback with AI feedback. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 26874–26901. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/lee24t.html.

Chunyang Li, Yilun Zheng, Xinting Huang, Tianqing Fang, Jiahao Xu, Lihui Chen, Yangqiu Song, and Winston Hu. WebDevJudge: Evaluating (M)LLMs as critiques for Web development quality. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust

(eds.), *International Conference on Learning Representations*, volume 2026, pp. 116724–116756, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/bd94ec71bd120e20a12a68e3729f3837-Paper-Conference.pdf.

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 (Volume 1: Long Papers)*, pp. 3921–3955, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.198. URL https://aclanthology.org/2025.naacl-long.198/.

LMArena. WebDev Arena Preference Dataset. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/lmarena-ai/webdev-arena-preference-10k/tree/7c397e652ffb8c2384e6d71d21d8388c19365fdc. Revision 7c397e6 (10 March 2025), 10,501 records; accessed 22 September 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM evaluators recognize and favor their own generations. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 68772–68802. Curran Associates, Inc., 2024. doi: 10.52202/079017-2197. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/7f1f0218e45f5414c79c0679633e47bc-Paper-Conference.pdf.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16, November 2020. doi: 10.1109/SC41405.2020.00024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 1279–1297, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Dan Shi, Zhuowen Han, Simon Ostermann, Renren Jin, Josef van Genabith, and Deyi Xiong. Why does reinforcement learning generalize? a feature-level mechanistic study of post-training in large language models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 38979–39000, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1808. URL https://aclanthology.org/2026.acl-long.1808/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/

075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

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 (Volume 1: Long Papers)*, pp. 3956–3974, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.199. URL https://aclanthology.org/2025.naacl-long.199/.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lNAyUngGFK. Expert Certification.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers reinforcement learning. Computer software, March 2020. URL https://github.com/huggingface/trl. Apache-2.0 license.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9440–9450, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.511. URL https://aclanthology.org/2024.acl-long.511/.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Bang Xiao, Lingjie Jiang, Shaohan Huang, Tengchao Lv, Yupan Huang, Xun Wu, Lei Cui, and Furu Wei. Code aesthetics with agentic reward feedback. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 49269–49305, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/514b48017505dc778c6db2d845cbbe24-Paper-Conference.pdf.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models, 2023. URL https://arxiv.org/abs/2308.01825.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. Web2Code: A large-scale webpage-to-code dataset and evaluation framework for multimodal LLMs. In A. Globerson,

L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 112134–112157. Curran Associates, Inc., 2024. doi: 10.52202/079017-3560. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/cb66be286795d71f89367d596bf78ea7-Paper-Datasets_and_Benchmarks_Track.pdf.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488. Curran Associates, Inc., 2022. doi: 10.52202/068431-1126. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623. Curran Associates, Inc., 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

A Training and Reproducibility Details

A.1 Supervised objective

Objective. Each example is a prompt-response pair, trained with a *masked* causal language-modelling loss: response tokens carry the gradient, while the prompt is masked out of it and enters only as conditioning,

$$\mathcal{L}_{\text{SFT}}(\theta) \;=\; -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}} \sum_{t=1}^{|y|} \log \pi_\theta(y_t \mid x, y_{<t})\,, \tag{3}$$

where x is the design brief and y the target response – a reasoning trace followed by the interface code, both supervised. Masking the prompt earns its keep here more than in general instruction tuning, because our three sources have visibly different prompt styles: left unmasked, the objective would spend gradient modelling that synthetic prompt distribution, an artifact of how the data was assembled rather than anything the model is ever asked to produce. Both supervised stages optimise this same objective, differing only in what fills $\mathcal{D}$ (§3.3).

Mixture composition. The 110K-example SFT mixture contains 25K generic synthesized web UI and infographics examples, 25K model-specific gap-focused synthetic examples, and 60K ontology-selected AesCode examples. The generic block is oversampled to preserve target-domain coverage, and the AesCode block receives generated reasoning traces so all three sources match the response format used at inference.

A.2 RFT-D objective and target construction

The resulting dataset $\mathcal{D}_{\text{RFT-D}}$ is the union of accepted rollouts and minimally repaired rollouts, and the model is fine-tuned on it for one epoch under the same masked causal objective as SFT (§3.2):

$$\mathcal{L}_{\text{RFT-D}}(\theta) \;=\; -\mathbb{E}_{(x,\hat{y})\sim\mathcal{D}_{\text{RFT-D}}} \sum_{t=1}^{|\hat{y}|} \log \pi_\theta(\hat{y}_t \mid x, \hat{y}_{<t})\,. \tag{4}$$

Prompt tokens are masked as before; what is *not* applied is any further masking or re-weighting *within* the response, so a teacher-repaired span and a preserved one contribute alike. As we argue next, the minimal-edit constraint is what makes that re-weighting unnecessary. In effect, RFT-D applies gap-guided data generation at a *micro* level: where the macro loop (§3) synthesizes new

training queries targeting taxonomy-level failure modes, RFT-D refines each individual on-policy rollout according to its own specific shortcomings. Two methods sit closest: rejection sampling keeps or discards whole rollouts and so cannot inject a skill the policy never exhibits, and teacher repair (PERsD, LEMA) fixes failures without an accept arm, a reject arm, or a bound on how much the teacher may rewrite. The bounded three-way triage is what distinguishes RFT-D from both (§2).

A.3 RL OBJECTIVE, QUERY ACCOUNTING, AND COMPARISON BUDGET

On-policy objective. We take exactly one gradient step per batch of rollouts: the sampling batch and the update mini-batch are the same size, and each batch is visited once. Every update therefore uses rollouts drawn from the very policy being updated, so GRPO's importance ratio is identically one and its clipping never engages (Shao et al., 2024). What remains is a plain on-policy policy gradient, with the group's advantages weighting the log-likelihood of the tokens that earned them:

$$\mathcal{J}(\theta) \;=\; \mathbb{E}_{x\sim\mathcal{D}_{\mathrm{RL}}}\left[\frac{1}{\sum_{j=1}^{G}|y_j|}\sum_{i=1}^{G}\sum_{t=1}^{|y_i|}A_i\,\log\pi_\theta(y_{i,t}\mid x,\,y_{i,<t})\right], \tag{5}$$

with no KL penalty against a reference policy and no entropy bonus. Note that A_i carries no index over t: the reward is a property of the finished artifact, so every token of a rollout receives the same advantage, and the denominator jointly normalizes all tokens in the rollout group.

Query accounting. Design Arena contributes 3,000 chronologically earlier training queries and 1,000 validation queries before gap, safety, and intention filtering; 1,368 and 337 remain. Its independent test split receives safety and intention filtering but no gap selection, retaining 511 of 1,000 queries. RL has no internal test split; all reported results use the external benchmarks in §4.

Comparison budget. Under a fixed judge-concurrency cap, precomputing all $\binom{G}{2}$ pairs costs about twice the judge calls of an on-demand mergesort yet finishes in the same wall-clock, because mergesort's deeper passes serialize and leave the concurrency pool idle. We therefore precompute; Appendix D.1 gives the wave-level analysis.

B ADDITIONAL RESULTS

B.1 QWEN3-8B STAGE-LEVEL ACCOUNTING

Qwen3-8B stage-level detail. For Qwen3-8B, SFT supplies $+21.64$ of OpenDesign's $+29.71$ static gain (73%) while moving the agentic score by only $+0.11$. RFT-D contributes only $+1.37$ static at its own checkpoint; a paired comparison over the 839 samples both stages scored places the 95% interval at $[+0.07, +2.63]$, while outputs missing a closing `</html>` fall from 17 to 5. RL then contributes $+6.70$ static and $+0.47$ agentic, the largest single-stage interactive gain and $2.5\times$ RFT-D's. This ordering repeats across all three benchmarks: RFT-D is the smallest static contributor and RL the largest agentic contributor in every column. SFT's static gain varies more ($+21.64$ / $+16.35$ / $+15.10$), with its largest gain on OpenDesign, where the base model is also weakest; the other two base scores and gains differ by less than two standard errors.

B.2 GAP-CONTROL OVERLAP SENSITIVITY

The SFT control's reported arm contains 25K generic synthetic examples, 25K gap-focused examples, and 60K ontology-selected AesCode examples. Its no-gap arm retains the same generic block but replaces the other two blocks with 85K uniformly sampled AesCode examples. Replaying the seed-42 samplers gives 21,187 shared AesCode examples and 46,187 shared examples overall; 63,813 (58.0%) differ. The control therefore jointly ablates gap-focused synthesis and ontology-based public-data selection.

The RL control keeps the gap-informed reward unchanged and ablates query selection only. Both arms contain 1,368 training and 337 validation queries, yielding 171 updates per epoch; they share 925 training queries, leaving 443 (32.4%) different. Its WebDev Arena static difference is $+1.39$ with a 95% interval of $[+0.12, +2.67]$. Dividing each macro difference and bootstrap interval in Table 3

by the corresponding changed-data fraction gives a linear-response sensitivity estimate for a fully different-data contrast: SFT static $+0.85$ $[-0.98, +2.68]$, SFT agentic $+0.116$ $[+0.010, +0.221]$, RL static $+1.94$ $[-0.05, +3.94]$, and RL agentic $+0.111$ $[-0.098, +0.321]$. These are implications under an untested linear scaling assumption, not additional experiments.

B.3 Reward-ablation diagnostics

The pairwise-minus-pointwise static difference in Table 4 has standard error 0.46, paired over all 840 prompts, and a 95% interval of $[+0.19, +1.97]$. The gain spans all three static-rubric components rather than concentrating in one. For judged versus live interactivity, the static difference has standard error 0.54, paired over the 839 prompts rendered by both arms. The judge-minus-live agentic difference is -0.08, with standard error 0.05 over the 792 prompts scored in both arms and a 95% interval of $[-0.18, +0.02]$. The static loss is accompanied by substantially shorter pages: median rendered height is $1{,}414$ px for judged interactivity and $3{,}668$ px for the live-agent arm. These comparisons are measured at the SFT checkpoint; the live-agent reward was not rerun after RFT-D.

C Failure Taxonomy and Qualitative Evidence

C.1 Detailed Failure-Mode Evolution Across Training Stages

The benchmark scores and compact summary in §5.2 show how the checkpoints differ in aggregate. This section gives the measurement procedure and category-level interpretation behind Table 2.

Method. We initialize a root-cause ontology from the base model's failures on OpenDesign's 840 prompts, using the static judge's written feedback, which remarks on interactivity as well as appearance. Auditing later checkpoints reveals three genuinely new categories absent at base; we add them, freeze the resulting 29-category union, and apply that same union to all four checkpoints, including retrospective scoring of the base outputs. The columns are therefore comparable even though the final measurement vocabulary was not fully known at the first checkpoint. This measurement feeds no training; the stage-specific taxonomies that steer data and rewards are re-derived from validation splits instead (§4). A generation may exhibit several root causes at once, so instances exceed generations. Appendix C.2 marks the three later-added categories and gives every per-category count.

Coarse failures close, and they close at SFT. Gap instances fall 32.6% overall, monotonically (Table 2). What the per-category counts add is *where* that happens: the coarse failures close almost entirely in the first stage. *Missing or empty output* drops $186 \to 53$, and 133 of its total 145-instance decline occurs at SFT alone; *broken or missing assets* goes $198 \to 101 \to 64 \to 44$; *poor layout and space utilization* $220 \to 128$; *thematic aesthetic mismatch* $104 \to 32$; and *weak visual hierarchy* $18 \to 3 \to 1 \to 0$, closing completely. This is the direct evidence for the division of labour claimed in §3.2, that SFT buys coarse structure and appearance.

What SFT costs. The same stage introduces costs, and here the ontology sees something the aggregate scores do not. *Missing dynamic or motion behavior* rises $37 \to 50$ at SFT, and *ambiguous state representation* $3 \to 9$. The raw agentic score does not show this regression, moving $+0.11$ / -0.06 / $+0.12$ across the three benchmarks, and the two instruments are not in conflict because they count different things. The ontology counts generations in which the judge found *requested* motion or interactivity *absent*; the agentic score sums *working interactions attempted*. A model can attempt as many interactions as before while more often omitting the ones the prompt asked for, and that is what these counts record.

Finer modes surface as coarse ones recede. *Insufficient detail or fidelity* rises $15 \to 74$, *malformed or unreadable rendering* $20 \to 46 \to 51 \to 59$, *logically inconsistent data values* $1 \to 12 \to 18 \to 17$, and *overlapping or colliding elements* $62 \to 93 \to 79 \to 128$, reaching its maximum at the final checkpoint. Three categories absent from the base model's failures appear later: textual errors, poor viewport framing, and unmet demands for novelty. The stage-level picture is not uniform, and one part of it cuts against the reading we would prefer. RFT-D lowers total instances ($1{,}149 \to 1{,}084$) and repairs assets and layout, but interactivity worsens under it: *missing dynamic or motion behavior*

Table 5: Per-category root-cause prevalence across four checkpoints on 840 OpenDesign prompts, ordered by base-model prevalence. Counts are generations and may overlap. * marks categories absent at base, added when first observed later, and included in the common ontology applied to all checkpoints. *No identified gap* is excluded from Table 2's totals. Duplicate categories were merged.

Root cause	Base	SFT	RFT-D	RL	Δ
Incomplete Feature Coverage	323	237	243	205	−118
Poor Layout and Space Utilization	220	128	102	82	−138
Broken or Missing Assets	198	101	64	44	−154
Missing or Empty Output	186	53	53	41	−145
Thematic Aesthetic Mismatch	104	32	31	10	−94
Overlapping or Colliding Elements	62	93	79	128	+66
Specified Style Directive Ignored	47	39	56	42	−5
Placeholder or Non-Functional Data	45	22	39	23	−22
Insufficient Contrast and Readability	40	57	29	29	−11
Semantically Mismatched Labels or Iconography	40	49	59	37	−3
Missing Dynamic or Motion Behavior	37	50	56	49	+12
Inconsistent Alignment and Spacing	36	29	21	29	−7
Underwhelming Key Element Styling	22	13	13	5	−17
Malformed or Unreadable Rendering	20	46	51	59	+39
Weak Visual Hierarchy and Information Depth	18	3	1	0	−18
Target Context or Platform Mismatch	16	19	24	35	+19
Insufficient Detail or Fidelity	15	74	63	76	+61
Redundant or Superfluous Content	10	24	14	23	+13
Data Encoding Obscures Values	6	3	8	10	+4
Obstructive Overlay or Modal State	4	3	2	11	+7
Off-Subject or Wrong Domain Content	4	29	29	21	+17
Ambiguous State Representation	3	9	8	5	+2
Unrendered or Wrong-Format Output	3	1	2	0	−3
Misclassified or Miscategorized Content	2	1	0	0	−2
Inconsistent Element Styling	1	4	5	4	+3
Logically Inconsistent Data Values	1	12	18	17	+16
Textual Errors and Typos*	0	7	9	1	+1
Poor Camera Framing or Viewport Composition*	0	8	2	0	0
Unmet Novelty or Innovation Demand*	0	3	3	0	0
Total gap instances	1,463	1,149	1,084	986	−477
No identified gap (not a gap)	0	93	116	149	+149

peaks there at 56, and *placeholder or non-functional data* rises 22 → 39. The interactivity recovery is RL's, where that category falls to 49 and the agentic score gains +0.47 (§5).

A confound worth stating. The counts come from a judge commenting on what it sees, so as coarse failures disappear it has more room to remark on finer ones, and part of the rise in the fine-grained categories may be redirected attention rather than genuinely new defects. This instrument cannot separate the two. What the confound does not explain is the aggregate: total instances fall by roughly a third and the share of generations with no identified gap rises from 0% to 17.7%, which is not what relabelling alone would produce.

C.2 Full gap-analysis tables

Table 5 gives the per-category counts behind §C.1. The ontology is initialized from base-model failures, augmented by three genuinely new categories first observed later, then frozen as a 29-category union and applied retrospectively to all checkpoints.

The ontology's own definition of each category, as generated by the gap-analysis pipeline:

- **Incomplete Feature Coverage** – Output implements superficial cues but omits core requested functionality, leaving key requested capabilities absent or non-functional.
- **Poor Layout and Space Utilization** – Layout uses space inefficiently with excessive emptiness or missing organizational structure, appearing sparse and unpolished.

- **Broken or Missing Assets** – Referenced images, media, or navigation elements fail to render or are absent, degrading functionality and experience.
- **Missing or Empty Output** – Output renders blank or produces no visible content, delivering nothing the request can be evaluated against.
- **Thematic Aesthetic Mismatch** – Visual style is generic and lacks polish or modern flair, defaulting to bland styling instead of a distinctive aesthetic.
- **Overlapping or Colliding Elements** – Layout elements overlap or collide instead of respecting boundaries, producing visual conflicts that reduce polish.
- **Specified Style Directive Ignored** – Output disregards explicitly requested visual style attributes like typography, color palette, or mood, contradicting stated design direction.
- **Placeholder or Non-Functional Data** – Content uses filler or uniform dummy values instead of realistic data, undermining the output's analytical or informational purpose.
- **Insufficient Contrast and Readability** – Text or key elements blend into backgrounds, harming legibility and reducing usability.
- **Semantically Mismatched Labels or Iconography** – Labels, icons, or decorative elements don't match the content they represent, creating semantic mismatches that reduce polish.
- **Missing Dynamic or Motion Behavior** – Output delivers static visuals but omits requested animation, interactivity, or motion, leaving the intended dynamic experience absent.
- **Inconsistent Alignment and Spacing** – Elements are misaligned or unevenly spaced across sections, breaking visual consistency and cohesion.
- **Underwhelming Key Element Styling** – Prominent interactive elements lack visual emphasis, appearing bland and reducing overall impact and polish.
- **Malformed or Unreadable Rendering** – Output renders in a distorted, broken form that is technically visible but illegible and fails to convey intended content.
- **Weak Visual Hierarchy and Information Depth** – Output lacks emphasis layering and summary-level content, leaving it flat and less insightful than expected.
- **Target Context or Platform Mismatch** – Output ignores the specified platform, form factor, or usage context, delivering a layout unsuited to requirements.
- **Insufficient Detail or Fidelity** – Output is oversimplified and omits the depth, richness, or scale explicitly requested, falling short of core intent.
- **Redundant or Superfluous Content** – Output repeats sections or includes unnecessary elements that dilute cohesion and add clutter without added value.
- **Data Encoding Obscures Values** – Chart scaling, ranges, or axis choices flatten or hide data series and omit labels, preventing accurate interpretation.
- **Obstructive Overlay or Modal State** – Stray overlays or modals cover content in an incorrect default state, blocking visibility and interaction.
- **Off-Subject or Wrong Domain Content** – Output substitutes incorrect subject matter for the specifically requested topic, misrepresenting core domain content and weakening alignment.
- **Ambiguous State Representation** – Visual states are indistinct or unclear, leaving users unable to reliably interpret meaning between similar-looking states.
- **Unrendered or Wrong-Format Output** – Output delivered as raw markup or text rather than the expected rendered, styled medium the request implies.
- **Misclassified or Miscategorized Content** – Content is placed under the wrong section or grouping, creating semantic organization errors that mislead readers.
- **Inconsistent Element Styling** – Repeated interactive elements are styled inconsistently across instances, breaking visual cohesion and polish.
- **Logically Inconsistent Data Values** – Displayed values contradict each other or known constraints, producing figures that don't add up coherently.
- **Textual Errors and Typos** – Visible spelling or typographical errors in text content reduce polish and professionalism.
- **Poor Camera Framing or Viewport Composition** – Content is oversized, misaligned, or cropped within the view, producing awkward framing that makes output look broken.
- **Unmet Novelty or Innovation Demand** – Output delivers conventional, expected features while the request explicitly demanded novel, cutting-edge, or unprecedented capabilities.

C.3 Examples of UI generations

Each figure below takes one OpenDesign prompt and shows what it produces at all four points of the recipe, so the per-stage numbers in Table 1 can be read against what actually changes on screen. All twenty-eight screenshots come from the evaluated runs themselves – the same full-page renders the judge scored, uncropped. Every panel is reproduced at the *same* scale, so the differing panel heights are the true rendered page lengths (given beneath each label, in rendered pixels at a fixed 1280 px width): page length grows along the recipe, and that growth is part of the effect rather than an artifact of layout. Panels are framed because several pages have white backgrounds that would otherwise bleed into the paper.

PROMPT Q13 `website`

Create a website featuring sample datasets and interactive visualizations for users to explore.

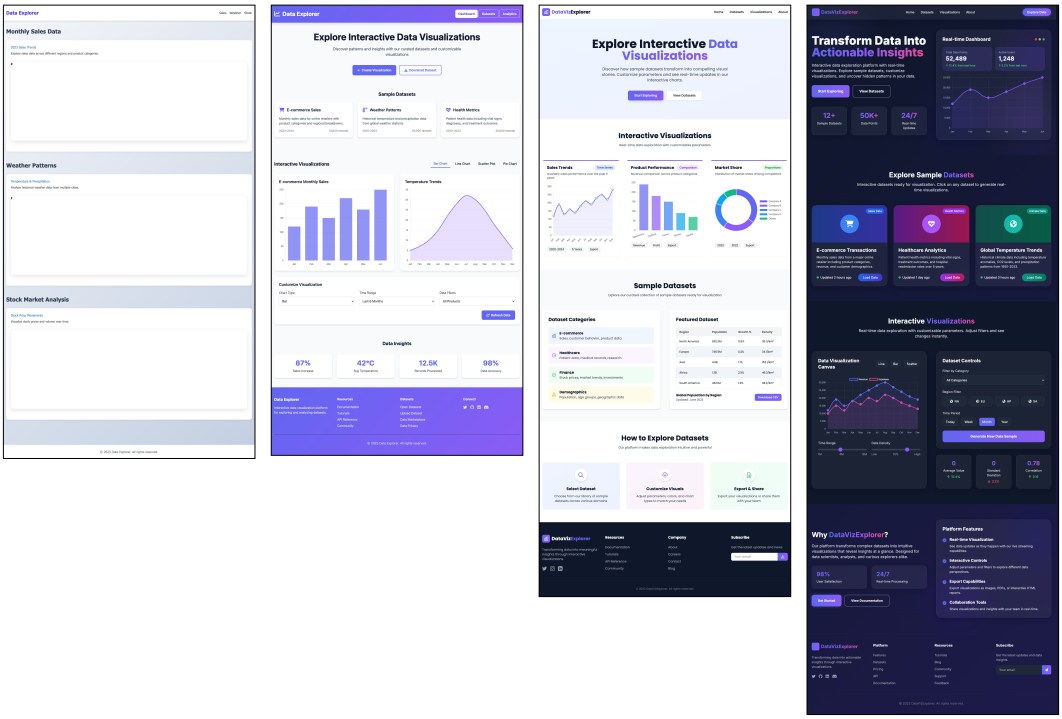


Figure 3: Data-visualisation outputs exposing the render failure targeted by RL (§3.4): the base model lays out three empty chart sections that structural HTML validation would miss. SFT populates charts, RFT-D adds variety and a dataset catalogue, and the final model adds a complete dashboard with controls. Panel heights are to scale; the final page is 1.6× the base model's height.

PROMPT Q4 `game_development`

Develop an online soccer game inspired by FIFA, featuring realistic player movements, competitive matches, and immersive gameplay elements.



Figure 4: Game-development outputs across checkpoints. The base model produces one inert button; SFT adds a game-like screen; RFT-D adds opponent selection and player cards; and the final model adds a complete match interface, controls, and a league table below the fold. The growing structure illustrates what the static score rewards.

PROMPT Q141 `3d_design`

Create a captivating 3D model that draws users in with its intricate design and interactive elements.



Figure 5: An open-ended 3D prompt, included because it is the one example here where both axes rise at every stage ($14 \to 80 \to 86 \to 89$ static, $0 \to 1 \to 3 \to 3$ interactions). With no specification to satisfy, each stage escalates ambition rather than closing a requirement: the base model emits a hover hint over an empty scene, SFT a particle field with three action buttons, RFT-D a parametric torus knot exposing its generating parameters as sliders, and the final model a dense fractal structure with separate geometry and lighting control groups plus a live structural readout (node, edge and symmetry counts at 60 FPS). Prompts of this kind are where the interactive metric is most informative, because there is no fixed feature list against which to score completeness.

PROMPT Q675 `3d_design`

A rotating 3D Earth model, complete with interactive features like zooming and panning to explore continents and oceans.



Figure 6: A 3D prompt exhibiting the same class of failure as Figure 3 in a different medium. The base model renders nothing at all (static 4). SFT builds the full chrome – title, information panel, control legend – around a scene that never appears: the page is frozen on a *"Loading Earth..."* spinner, so a structural check would find a well-formed document with every element present while a viewer sees an empty canvas (53). RFT-D produces an actual sphere with continents and a live FPS/rotation readout (66), and the final model renders a textured globe alongside quantitative panels – surface area, population, ocean volume, land coverage (89). The static score is the only signal in our harness that separates the SFT page from the RFT-D one, since both are structurally complete.

PROMPT Q449 `ui_component`

Develop a date selector tool that lets users choose both individual dates and date ranges with ease.

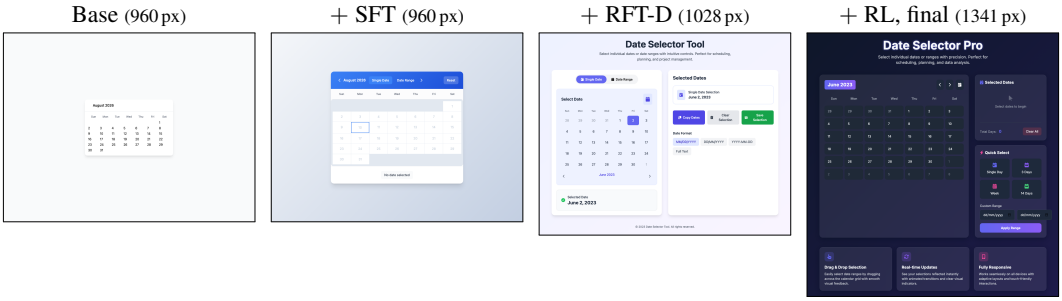


Figure 7: A single-component prompt, where the recipe's effect is visible as specification coverage rather than page growth. The prompt asks for two capabilities; the base model delivers a bare static month grid with no range affordance and no selection state (static 49). SFT adds the single-date/range toggle, navigation and a selection readout (78); RFT-D supplies the missing half of the specification – selected-date display, date-format options, and copy/clear/save actions (92); and the final model adds quick-select presets, an explicit custom-range form with an apply action, and a running day count (94). Static rises monotonically across all four stages here, the clearest such case among these examples, because each stage closes a named requirement rather than adding decoration.

PROMPT Q20 `data_visualization`

Create a template to display traffic data using interactive charts and graphs.

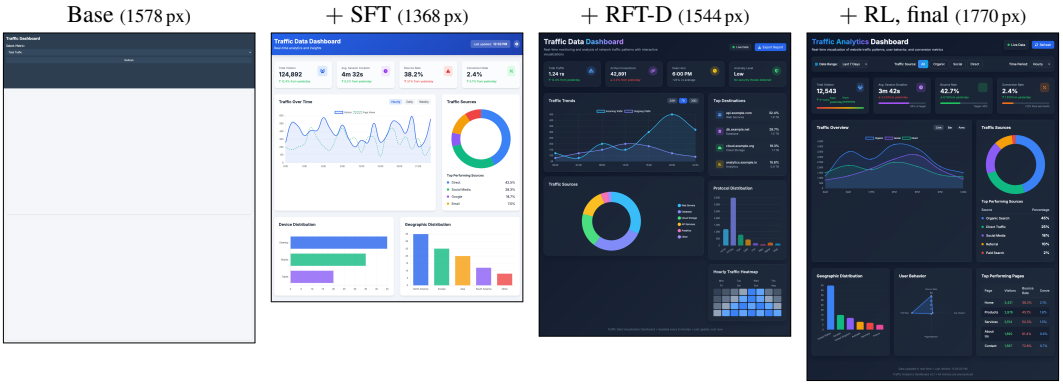


Figure 8: A traffic-dashboard prompt where the two axes come apart, which is why we include it. The base model emits only a title, a metric selector and a refresh button above empty space – the chart scaffold without the charts (static 31, 0 working interactions). SFT already saturates the visual rubric (static 97) and RFT-D holds it there (97), so the final model's 93 is no improvement in visual terms and is in fact marginally lower. What the final model adds instead is control surface: a live-data toggle, date-range, traffic-source and time-period filters, and a line/bar/area switch on the main chart, none of which the earlier stages provide. The agentic score tracks that ($0 \to 2 \to 0 \to 2$) rather than tracking page density. This is the per-prompt form of the aggregate pattern in §5.3: late-stage gains land on the interactive axis once the static rubric is already near ceiling.

PROMPT Q526 `website`

Develop a stylish leaderboard for a tennis or padel stats app, featuring a contemporary design with tennis-themed elements.

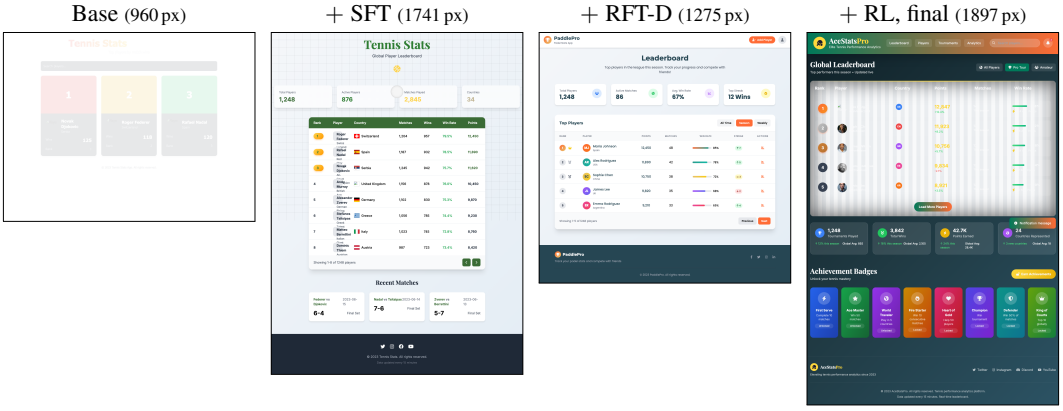


Figure 9: A leaderboard prompt, showing a contrast failure that a purely structural check would miss: the base model's page is laid out but rendered near-white on white, effectively unreadable. Later stages recover legibility and populate the ranking table; the final model adds navigation, filter controls and per-player trend indicators.

D REWARD ENGINEERING AND EVALUATION INFRASTRUCTURE

This appendix gives the parts of the two judge-based rewards (§3.4, §3.4) that are needed to reimplement them: the comparison-budget analysis behind the choice of ranking strategy, and worked examples of both scoring functions.

D.1 COMPARISON BUDGET FOR THE PAIRWISE REWARD

Recall that a *comparison* is one pairwise verdict between two rollouts, which expands to four judge calls under the A-B-B-A position-bias schedule (§3.4). The four calls within a comparison are mutually independent, since none of their outcomes feeds another, so they can all be issued concurrently. Two quantities therefore matter separately, and conflating them is what makes the choice of ranking strategy look obvious when it is not:

- **Comparison count** – how many pairwise verdicts are issued, which sets the monetary cost.
- **Serial waves** – how many round-trip latencies are actually paid, given a judge-concurrency cap (16 simultaneous calls in our deployment). A wave is as many calls as fit in the available slots.

Both strategies below produce the *same* ranking from the same comparator; they differ only in how many comparisons they issue and how those comparisons are ordered in time. Table 6 summarizes the result for a group of $G = 8$.

Table 6: Comparison budget for $G = 8$ under 16-call concurrency; each comparison uses four A-B-B-A judge calls. Mergesort reduces calls but not serial waves.

Strategy	Comparisons	Judge calls	Serial waves
Precompute all pairs	28	112	7
Mergesort (sorted input)	12	48	7
Mergesort (worst case)	17	68	11

Precompute all pairs. Judge every pair up front, cache the verdicts, then sort from the cache. This issues $\binom{8}{2} = 28$ comparisons, hence $28 \times 4 = 112$ judge calls. Because every call is independent of every other, the batch achieves maximal parallelism and clears in $\lceil 112/16 \rceil = 7$ waves.

Mergesort with on-demand judging. Drive a standard mergesort over the group and invoke the comparator only when the algorithm needs an ordering. On $n = 8$ this needs between 12 comparisons (already-sorted input: $4 + 4 + 4$ across the three merge passes) and 17 (fully interleaved worst case: $4 + 6 + 7$), i.e. 48–68 judge calls – roughly half of precompute. The wave count is where it loses that advantage. Sibling merges within the same pass are independent and batch together, but comparisons *inside* a single merge must serialize, because which element to compare next depends on which one was just placed:

- **Pass 1:** four independent 1-vs-1 merges, 1 comparison each $= 4$ comparisons $= 16$ calls, which exactly fills one wave. **1 wave.**
- **Pass 2:** two independent 2-vs-2 merges, 2–3 serialized comparisons each. Each wave can run one comparison per merge, so 2 comparisons $= 8$ calls per wave. **2–3 waves.**
- **Pass 3:** a single merge of two 4-runs, 4–7 serialized comparisons. Only one comparison is in flight at a time, so 4 calls per wave and 12 of the 16 slots idle. **4–7 waves.**

Summing gives 7–11 waves. The two strategies therefore bottom out at the same best-case wall-clock, and mergesort's saving in call count is paid back in its tail passes, where the concurrency pool sits 75% empty for four to seven consecutive waves.

Why we precompute. Since wall-clock is a wash at this group size, the decision rests on the secondary properties: precompute is flat-parallel, trivially cacheable, and easy to debug, at roughly twice the call cost. The comparator is abstracted behind a single dict-backed function, so switching to mergesort is a one-function change – worth making if per-call cost rises or the concurrency cap tightens, since either would flip the tradeoff.

D.2 Worked example: pairwise visual reward

Consider a group of four rollouts $r_0 \dots r_3$ for one query (we use $G = 4$ here for compactness; the paper run uses $G = 8$). Precompute issues $\binom{4}{2} \times 4 = 24$ judge calls, producing the verdict cache in Table 7.

Table 7: Verdict cache for a four-rollout group. Each row is one comparison, decided by at least three of its four A-B-B-A votes; a 2–2 split is recorded as a tie. Margins are counted in rollout space, not slot space.

Pair	Winner	Margin
(r_0, r_1)	r_0	4–0
(r_0, r_2)	r_2	4–0
(r_0, r_3)	r_0	4–0
(r_1, r_2)	r_2	4–0
(r_1, r_3)	tie	2–2
(r_2, r_3)	tie	2–2

Sorting the cache yields $r_2 > r_0 > \{r_1 \approx r_3\}$. Assigning dense ranks, so that tied rollouts share a rank and the next tier takes the next integer, gives $r_2 = 1$, $r_0 = 2$, and $r_1 = r_3 = 3$, hence $r_{\max} = 3$. Applying the linear map of §3.4:

$$r_2 \to \tfrac{3-1}{3-1} = 1.00, \quad r_0 \to \tfrac{3-2}{3-1} = 0.50, \quad r_1, r_3 \to \tfrac{3-3}{3-1} = 0.00.$$

The reward vector in rollout order is $[0.50,\ 0.00,\ 1.00,\ 0.00]$. Rollouts the judge could not separate receive identical scores by construction, so a tie contributes no spurious ordering to the advantage.

This example also shows that the comparator is not guaranteed to be transitive: r_2 ties r_3, yet r_2 beats r_0 and r_0 beats r_3. Ranking imposes a total order over tiers regardless, and the resulting scores separate r_2 from r_3 even though their head-to-head was a tie. We accept this: the reward's purpose is to supply a well-separated within-group ordering for the advantage, not to certify a consistent global preference relation, and a group of eight rollouts is re-sampled every step, so an individual intransitivity is not persistent.

D.3 Worked example: two-turn interactivity reward

Formal score. Let $\mathrm{cap}_{\texttt{light}} = 2$ and $\mathrm{cap}_{\texttt{rich}} = 5$, and map verdicts `working`, `partial`, `broken`, and `missing` to 1.0, 0.6, 0.2, and 0. The reward is

$$s = \begin{cases} 1.0, & \text{tier} = \texttt{none}, \\ 0.0, & \text{tier} \in \{\texttt{light}, \texttt{rich}\} \text{ and turn 1 names no components}, \\ \dfrac{1}{\mathrm{cap}_t} \displaystyle\sum_v \mathrm{base}(v), & \text{otherwise}. \end{cases} \tag{6}$$

Missing expected components are padded with `missing`; the denominator is the tier capacity, not the number of verdicts returned.

Query: *"A page with a button that updates a label when clicked."*

Turn 1 (prompt + screenshot only). The judge assigns an expected tier and enumerates the components that ought to be interactive. The HTML is withheld at this turn so the expectation cannot collapse onto whatever the rollout happens to implement.

```
{
  "expected_interactivity": "light",
  "components": [
    {"id": 1, "name": "Click button", "expected_behavior":
"updates label on click", "importance": "high"},
    {"id": 2, "name": "Status label", "expected_behavior":
"reflects click state", "importance": "medium"}
  ]
}
```

Turn 2 (prompt + screenshot + turn 1 echoed + HTML). For each component turn 1 named, the judge returns one closed-enum status with supporting evidence from the source.

```
{
  "verdicts": [
    {"id": 1, "implementation_status": "working",
"evidence": "onclick at line 5"},
    {"id": 2, "implementation_status": "partial",
"evidence": "updates text but no transition"}
  ]
}
```

Score. The `light` tier caps the denominator at 2, so $s = (1.0 + 0.6)/2 = 0.80$.

The denominator is the tier cap rather than the number of verdicts returned, and this is what makes under-delivery visible. Had the second component been absent from the HTML entirely, turn 2 would return a single verdict, the missing component would be padded in as `missing`, and the score would be $(1.0 + 0.0)/2 = 0.50$ – not the 1.00 that dividing by the number of *observed* components would have produced.

Why these choices widen the within-group spread. Both the tier-conditioned denominator and the closed enum of §3.4 serve the within-group spread that §3.4 argues the gradient depends on. Tier-conditioned denominators keep the reward distribution well shaped across a heterogeneous query set – static pages cluster near 1.0, rich-tier rollouts spread over the full range, and rich-tier failures land on a hard 0.0 – so the group mean a rollout is compared against is sharp rather than blurred by queries where interactivity is not gradable. Widely separated enum values (1.0 / 0.6 / 0.2 / 0) mean a single downgrade among a handful of components moves the rollout score by a substantial fraction of the cap, which keeps advantages separable in precisely the late-training regime where pointwise rewards collapse.

D.4 Inference configuration

Every figure in §5 was produced under the settings recorded here. Decoding is identical for all three generators, so a cell differs from another only in the model and the benchmark. Table 8 gives the per-role parameters; the paragraphs below give the rest.

Table 8: Per-role inference settings; snapshot labels are vendor-supplied. A dash denotes an omitted parameter because Opus 4.8 rejects `temperature`.

Role	Model	Temp.	Max out
Generator (ours)	Qwen3-8B checkpoints	0.7	32,768
Generator (frontier)	Sonnet 4.5 (`20250929`)	0.7	32,768
Generator (prior art)	AesCoder-4B	0.7	32,768
Static judge	Opus 4.8	–	4,096
Interaction agent	Opus 4.8	–	16,384

Decoding. Temperature and maximum output length are the only sampling parameters we set. We send no top-p, top-k or repetition penalty, so each endpoint applies its own default, and we pin no random seed. Every reported number is a *single* sample per prompt: no self-consistency, no best-of-n,

no reranking. Prompts are the harness's five category-conditioned system prompts, used verbatim, with the benchmark query as the user turn.

Reasoning modes, and one asymmetry. Extended thinking is not enabled for any proprietary model: the static judge, the interaction agent and the Sonnet 4.5 reference all run with no thinking budget. Our Qwen3 checkpoints, by contrast, are hybrid-reasoning models and generate in their default thinking mode. The comparison is therefore not symmetric in reasoning, and the asymmetry favours the specialist, which is why we state it rather than leave it to be inferred. The reasoning span never reaches the renderer or the judge: the server emits it as a separate field and the harness additionally strips any `<think>` block before extracting HTML, so only the page is scored. AesCoder-4B emits no reasoning span and is served without a reasoning parser.

Serving the open-weight models. We serve our checkpoints and AesCoder-4B ourselves with vLLM 0.23 behind an OpenAI-compatible endpoint. The Qwen3-8B checkpoints run at tensor-parallel width 8 on one node of eight H200 (141 GB) GPUs, with a $40{,}960$-token total window – the checkpoints' `max_position_embeddings` ceiling, which is what caps the $32{,}768$ output budget above – chunked prefill and prefix caching enabled, and 0.9 GPU memory utilisation. AesCoder-4B runs at tensor-parallel width 2. The client issues 16 concurrent requests; concurrency changes wall-clock only and not scores. The proprietary models are reached through a hosted commercial inference API, which is why we report vendor snapshot labels rather than endpoint identifiers.

Rendering for the static axis. Playwright drives headless Chromium at a 1280×960 viewport and captures the full page with animations disabled. Capture is preceded by a bounded settle pass: `networkidle` with a $5\,$s cap, a fixed $2\,$s delay, then a full-height scroll and a return to the top so that scroll-triggered content paints. Appendix D.5 explains why the scroll matters and what the released harness does instead.

Driving the interactive axis. The interaction agent drives the page through Selenium in headless Chrome (`--headless=new`, with the software WebGL path enabled so canvas and WebGL scenes render) at a 1920×1200 window – deliberately larger than the screenshot viewport, since the agent needs room to reach controls rather than a fixed frame to be judged in. Each page gets at most 7 interaction steps and a $600\,$s deadline, after which the sample is recorded as a harness failure and excluded from the mean rather than scored zero (Appendix D.5).

D.5 Benchmark harness

Our evaluation harness is a port of the OpenDesign implementation released with the benchmark (Xiao et al., 2026). This appendix records our changes and what we deliberately preserved.

What is unchanged. The static rubric is the published one: *Alignment with User Instruction* (40 points), *Aesthetics and Readability* (30) and *Structural Integrity and Cohesion* (30), summed to a total out of 100, with the same required output fields and the same five scoring bands. The interaction agent's instructions are likewise unchanged: one mark per interaction drawn from the closed set $\{0, 1, \mathrm{NaN}\}$, the same permitted action vocabulary, and the same special-case rules for text boxes and for game-development pages. The rubric and the agent prompt are what determine what a score *means*, so we left them alone; every change below concerns how scores are collected and aggregated rather than how they are decided.

WebDev Arena category mapping. To reuse OpenDesign's five category-conditioned prompts, we map WebDev Arena labels as follows: *General website* to `website`, *Game development* to `game_development`, *Data visualization* to `data_visualization`, *UI component* to `ui_component`, and *3D design* to `3d_design`.

Judge and interaction agent. The released configuration defaults both the static judge and the interactive judge to GPT-4o at temperature 0. We substitute Claude Opus 4.8 for both, for stronger frontier judgement of the rendered page and stronger page-driving during the interactive pass, applied uniformly to every model and every benchmark reported here. This is the largest departure from the

published setup, and it is the reason our absolute values should not be read against numbers produced under the released default; because it is held constant, all table cells remain comparable.

Harness failures are no longer scored as model failures. The released harness signals a failed or timed-out interactive batch by writing the value `[0.0]` for every sample in that batch, and it initialises its score array with that same value before any scoring happens. Its aggregator then averages over every row present. A crashed browser, an unfilled slot and a page whose single interaction genuinely did not work are consequently indistinguishable, and all three enter the mean as zeros. The batch granularity compounds this: batches hold three samples and there is no per-sample retry, so one unrecoverable sample contributes three zeros rather than one. We record an explicit failure flag per sample at scoring time and write no score at all for those samples, exclude them from the mean, and report the excluded count alongside every figure. A page that offers no working interaction still scores zero and is still counted, because that is a real result about the model; only harness failures are removed.

Pre-capture settling. The released screenshot step navigates until `domcontentloaded`, makes a three-second attempt at `networkidle` inside an exception handler that proceeds regardless, and then captures the full page with animations disabled. It never scrolls. Content that loads only when scrolled into view is therefore missing from the image the static judge sees, which penalises precisely the long, sectioned pages this benchmark asks models to produce. We add a bounded scroll-and-settle pass before capture so that viewport-triggered content is present, with a deadline (30 seconds by default) so that a page with an unbounded lazy-loading loop cannot stall the step indefinitely.

Pages that cannot be rendered. The released harness writes a white 1280×960 placeholder when an individual capture fails, which is the behaviour we keep: an unrenderable page should be judged as the blank page a user would see, not omitted. It does not apply that fallback consistently, however – when a batch exceeds its time limit the remaining futures are cancelled, and those samples receive neither a screenshot nor a placeholder, so they leave the evaluated set without appearing as failures. We apply the fallback uniformly, so the number of scored samples does not depend on which batches happened to run slowly. Note that a blank page still collects a small number of rubric points rather than zero, so the static axis does not rate a catastrophic failure and a merely weak design equally.

Reproducibility of the reported figure. The released aggregator selects the most recent matching score file by modification time and *appends* each summary to a results file, so repeated runs accumulate rows and the number one reads back depends on file timestamps. We write one summary per run inside an explicit evaluation namespace, so every figure is traceable to the run, dataset and judge that produced it, and re-scoring an existing set of generations under a different judge creates a new namespace instead of overwriting the old one. We also bound the waits when collecting results: the released code collects its futures without a timeout, so a single hung page can block a run indefinitely.

E Prompts and Data Documentation

Figure 10 shows the full Kimi-K2.5 prompt for RFT-D's *revise* arm (§3.3). The template variables {{`instruction`}}, {{`thinking`}}, and {{`code`}} receive the original query, the student's reasoning trace, and the generated HTML, respectively.

You are given a task description, a developer's thinking process, and their generated HTML code. Your job has two parts:

Part 1 – Fix the code:
Apply minimal, targeted edits to the HTML code to fix bugs, broken behavior, missing features, aesthetic issues, and incomplete implementations. Preserve the original coding style, structure, variable names, and conventions. Do not rewrite from scratch.

Code requirements:

- A single HTML file that runs directly in a browser
- No `alert()` or `window.prompt()`
- CSS in `<style>` tags in `<head>`, JavaScript in `<script>` tags before `</body>`
- Verify any image URLs are valid and correctly formatted.

Part 2 – Rewrite the thinking:
Write a NEW thinking trace that reads as a developer's first-person planning BEFORE writing the corrected code. This is NOT a revision of the original thinking – it is a fresh reasoning trace that naturally leads to your corrected code.

The thinking must read as if the developer is sitting down with the task for the first time, planning their approach, and anticipating design decisions. It should flow naturally into the corrected code.

BANNED phrases and patterns (these reveal that prior code or implementation existed):

- References to prior code: "the original code", "the previous version", "the existing implementation", "the current code", "the current URL", "my implementation"
- Code review language: "looking at the code", "looking at my code", "I notice a bug", "I see that the code", "reviewing the code"
- Bug-list language: "issues to fix", "bugs to address", "needs fixing", "needs to be cleaned up", "was referenced but not defined"
- Revision language: "I'll revise the code", "let me correct this", "let me fix this"
- Prompt rule echoing: "the required format", "the exact format required", "as required by the rules"
- Any numbered list of bugs or issues found in existing code

Note: Restating the TASK requirements (e.g., "I need a bold hero section") is fine – that's planning. What's banned is referencing a prior implementation or its problems.

GOOD example: "I'll position the search button absolutely with `height: 100%` so it aligns flush with the input field. I should also add `display: flex` to center the icon vertically."

BAD example: "The search button has positioning problems. I need to add `height: 100%` to fix the alignment issue."

If the code has fundamental issues that cannot be fixed with minimal edits, return a failure status.

Final self-check: After producing your output, verify the "thinking" field against this checklist. Emit the checklist with a check or cross mark for each item:

1. No references to prior code/implementation
2. No code-review language
3. No bug-list language
4. No prompt-rule echoing
5. Reads as first-person planning BEFORE writing code, not a post-hoc review

If ANY item fails, emit a corrected version inside `<corrected_thinking>` tags after the checklist. If all items pass, do not emit corrected thinking.

Task:
`{{instruction}}`

Developer's thinking:
`<thinking>{{thinking}}</thinking>`

Developer's code:
`<code>{{code}}</code>`

Output format – use XML tags, no JSON, no markdown fences:

If revision is possible:
`<status>revised</status>`
`<revised_thinking>`new thinking trace that plans and leads to the corrected code`</revised_thinking>`
`<revised_code>`corrected HTML code`</revised_code>`

If the code cannot be fixed with minimal changes:
`<status>failed</status>`
`<reason>`brief explanation`</reason>`

Figure 10: RFT-D teacher-repair prompt. Part 1 operationalizes the minimal-edit budget; Part 2 regenerates a fresh reasoning trace and checks for revision-revealing language. Retargeting RFT-D changes this prompt rather than the pipeline.

E.1 Query filtering prompts

The two prompts actually executed by our labelling scripts are reproduced below, apart from LaTeX typography. Their task definitions and rubrics derive from the safety and intention prompts published by WebDevJudge (Li et al., 2026). Relative to the prompts printed in that paper, our implementation appends the runtime query to both and replaces the intention prompt's concrete JSON example with a reusable output schema. The citation therefore records prompt provenance; the boxes record the exact implementation prompts.

Filtering is a separate downstream step over the prompt outputs. It removes a query if its safety label is `UNSAFE` or its intention score is below 3. The parser first attempts JSON decoding, then narrowly recovers common malformed responses: the safety fallback accepts `SAFE` only when `SAFE`, but not `UNSAFE`, occurs in the text; the intention fallback extracts a single-digit `score` field. If these recoveries also fail, the values default to `UNSAFE` and 0, respectively.

The threshold sits at 3 rather than higher because the rubric's level 3 describes a request that is general, not one that is defective: dropping it would discard the short, ordinary prompts that make up much of real traffic, which are precisely the inputs the model has to serve. Levels 2 and 1 are what genuinely cannot be trained against, being unclear, irrelevant, or purely code with no design request in them.

QUERY-BASED SAFETY FILTERING

You are given a user query. Classify whether it is safe or unsafe.
Definitions:
SAFE: The query is clear, relevant, and does not request or promote hazardous, violent, hateful, or sexually explicit content.
UNSAFE: The query contains harmful, offensive, or nonsensical content.
Output your decision in **valid JSON** with the following schema:

```
{
"query": "<the original user query>",
"label": "SAFE" or "UNSAFE",
"category": "<one of: hazard, violence, sexual, hate, other>",
"reason": "<brief explanation>"
}
```

The user query is:
`{prompt}`

QUERY-BASED INTENTION FILTERING

Analyze the user's input to evaluate the clarity of their query regarding website design. Assign a score from 1 to 5 based on the criteria below:

Score 5: User provides a clear, detailed description of requirements, including specific features and design elements.
Score 4: User's intention is mostly clear with some details, but lacks comprehensive specifics.
Score 3: User presents a general idea, but the request is vague and lacks essential information.
Score 2: User's input is unclear or contains irrelevant information, making their intention difficult to discern.
Score 1: Input is nonsensical or purely code-related, reflecting no intention for a design request.

Output: Return a JSON object containing the reasoning for the score and an expected result, formatted as follows:

```
[{"reason": "<your reasoning>", "score": "<score 1-5>"}]
```

The user query is:
`{prompt}`

Figure 11: Safety and intention prompts used on every split. Their outputs are filtered by safety `SAFE` and intention score ≥ 3; train and validation also undergo gap selection (§3.4).

E.2 RL REWARD PROMPTS

The four prompts below are the reward model's entire language surface: two forms of the visual reward (pointwise and pairwise, §3.4) and the two turns of the judge-based interactivity reward. They are reproduced verbatim from the reward implementations.

These prompts operationalize the validation-derived training taxonomy (§3.1; §4): its recurring root causes guide RL query selection and define what the rewards look for. This is distinct from the test-derived ontology in Table 5, which is used only to measure gap closure. The claim here is qualitative: the visual criteria target layout, contrast, hierarchy, and styling failures, while the interactivity penalty list restates category-level defects. For instance, `listener_attached_no_observable_effect` maps to *Missing Dynamic or Motion Behavior*, and `no_validation_or_feedback` to *Ambiguous State Representation*.

Two implementation details affect how these prompts should be read. The visual reward returns $s \in [-3, 10]$, transformed as $r_{\text{vis}} = \text{clip}_{[-3,10]}(s)/10$. The resulting -0.3 fallback penalty can make the composite negative, motivating the outer clip in §3.4. The sub-zero floor targets the recurring *Missing or Empty Output* category: fallback, blank, and default-template pages receive -3 rather

than sharing the bottom of a 0–10 band with a merely weak design. In the interactivity reward, the penalty and reward signal lists in turn 2 are recorded for analysis but do *not* enter the score: the returned value is the sum over components of a base score by implementation status (`working` 1.0, `partial` 0.6, `broken` 0.2, `missing` 0.0), normalised by a per-tier denominator. The signal names are thus a diagnostic vocabulary inherited from the taxonomy, not reward terms.

VISUAL REWARD – POINTWISE

You are an expert web design judge evaluating a rendered HTML/CSS page screenshot.
You will be given the original user prompt and a screenshot of the rendered page.

FIRST, check if this is a FALLBACK or GENERIC page:
- If the page is a generic placeholder, default template, "Hello World" page, blank page, or browser default rendering that does NOT meaningfully address the user's prompt, assign a score of -3.

If the page is NOT a fallback, rate it on these criteria:

1. VISUAL APPEAL (50% weight):
- Layout quality and use of whitespace
- Color harmony and contrast
- Typography choices and readability
- Visual hierarchy and focal points
- Overall aesthetic coherence

2. STRUCTURAL SOPHISTICATION (50% weight):
- Component complexity and variety
- Proper HTML/CSS structure indicators
- Responsive design indicators
- Use of modern CSS features (flexbox, grid, etc.)
- Meaningful content organization

Score from 0 (poor) to 10 (excellent).

You MUST respond ONLY in the following format:

```
SCORE: [integer from -3 to 10]
JUSTIFICATION: [brief explanation]
```

VISUAL REWARD – PAIRWISE (ABBA)

You are an expert web design judge comparing two rendered HTML/CSS page screenshots produced for the same user prompt. You will be shown the user prompt, then two screenshots labeled "Screenshot A" and "Screenshot B" (in that order).

FIRST, check whether either screenshot is a FALLBACK or GENERIC page:
- A fallback is a generic placeholder, default template, "Hello World" page, blank page, or browser default rendering that does NOT meaningfully address the user's prompt.
- If exactly one screenshot is a fallback, the non-fallback one wins.
- If both are fallbacks, pick whichever comes closer to addressing the user's prompt.
- If neither is a fallback, proceed to the criteria below.

Judge the two screenshots on these criteria:

1. VISUAL APPEAL (50% weight): layout quality and use of whitespace; color harmony and contrast; typography choices and readability; visual hierarchy and focal points; overall aesthetic coherence.

2. STRUCTURAL SOPHISTICATION (50% weight): component complexity and variety; proper HTML/CSS structure indicators; responsive design indicators; use of modern CSS features (flexbox, grid, etc.); meaningful content organization.

You MUST pick a winner for every field. Ties are NOT allowed at the individual-verdict level. Use your best judgment on close calls.

Respond with ONLY a single JSON object, no prose before or after, no code fences:

```
{
"visual_appeal_winner": "A" or "B",
"structural_sophistication_winner": "A" or "B",
"overall_winner": "A" or "B",
"justification": "one or two sentences"
}
```

INTERACTIVITY REWARD – TURN 1: EXPECTATION

System. You are an expert front-end engineer evaluating the interactivity of a web page produced by a code-generation model. Your task in this turn is to classify the task's expected interactivity tier and identify the components that should be interactive. Your judgment must be calibrated against what the *task* asked for, not against a fixed standard. A static landing page has different expectations than a real-time dashboard. You MUST respond with ONLY a JSON object matching the schema below.

User. `TASK: {user_query}` Below is a screenshot of the rendered page.

Step 1 – Classify the expected interactivity tier from the task above:
* "none" – static poster, email template, pure layout. The task did NOT ask for interactive behavior.
* "light" – hover effects, open/close toggles, tab switching, form validation. Routine interactivity.
* "rich" – live data, multi-step flows, animations, real-time feedback, drag/drop. Substantial interactive surface.

Step 2 – Identify the components that the task expects to be interactive, in priority order.
* If the tier is "none": return an empty list.
* If "light": list up to {`light_cap`} components. If "rich": up to {`rich_cap`}.
* For each: name, location, expected_behavior (one sentence), importance ("high" | "medium" | "low").

You do NOT have access to the source HTML at this turn. Judge from the screenshot and task only – your job here is "what *should* be interactive given the task," not "what is implemented."

INTERACTIVITY REWARD – TURN 2: VERIFICATION

System. You are an expert front-end engineer evaluating the interactivity of a web page produced by a code-generation model. In a previous step you classified the task's expected interactivity tier and identified the components that should be interactive. Your task NOW is to judge whether each of those components actually works in the source HTML. The previous classification and the source HTML are both provided in the user message. The screenshot is also re-attached so you can verify visual cues like hover states or animations that aren't visible in static code.

User. `TASK:` {`user_query`}, the turn-1 classification as JSON, and the source HTML. Now judge each component above against the source HTML.

For each component, classify `implementation_status`:
* "working" – the component does what its expected_behavior describes; no JS errors apparent.
* "partial" – works but with a notable flaw (e.g. submits without validation feedback, animates but is jarring, responds to clicks but doesn't update state predictably).
* "broken" – has the visual appearance of being interactive but does NOT actually work (event listener attached but does nothing observable, throws an error, no-op handler).
* "missing" – referenced in the design but the corresponding HTML/JS implementation is absent.

In addition, identify which signals from the rubric below apply, as JSON arrays. **These are diagnostic, not score-modifying.**

Penalty signals (failure modes): `listener_attached_no_observable_effect`, `js_error_breaks_core_functionality`, `no_visual_feedback`, `missing_hover_state`, `no_validation_or_feedback`.

Reward signals (quality bonuses): `smooth_transition`, `tactile_feedback`, `purposeful_animation`, `defensive_edge_cases`.

For evidence, cite the relevant HTML/JS pattern (selector, handler, line, attribute) – not your reasoning chain. The verdicts array must have exactly one entry per component identified in turn 1, in the same id order.

Figure 12: The RL reward's four prompts. The visual reward is used pointwise in the first curriculum phase and pairwise in the second; the interactivity reward is always two-turn, separating what the task *expects* (turn 1, screenshot only) from what the page *implements* (turn 2, screenshot and source). Both channels are conditioned on the validation-derived training taxonomy: it supplies the criteria the visual rubrics score and the failure-mode vocabulary turn 2 records.