TOWARDS SELF-ADAPTIVE GENUI EXPERIENCES: AN APPROACH, BENCHMARK & EVALUATION SUITE

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative UI (GenUI) requires producing complete interactive experiences rather than static webpages: so the interfaces must remain functional, stateful, and usable across views and interactions. While frontier models generate visually plausible HTML, their outputs frequently fail under interaction, and these failures recur across tasks. We introduce SAGE-CREATE, a training-free framework that converts verified GUI repairs into reusable multimodal memory. Its Reflect–Refine–Remember loop infers a task specification, generates and executes candidate interfaces, and applies focused repairs that are admitted to memory only when before-and-after replay confirms that the failure is resolved and required behavior is preserved. Central to this loop is a multimodal memory that records how each rule fixed a concrete failure through code patches, screenshots, and interaction traces, together with a learning curriculum that governs rule creation, promotion, and retirement. To evaluate GenUI experiences, we introduce SAGE-BENCH, 300 tasks spanning an intent-based taxonomy of 11 experience families and 48 sub-families, and SAGE-EVAL, which measures perceptual alignment, visual layout, and data integrity alongside specification fidelity and execution reliability. SAGE-CREATE achieves 74.12 on SAGE-BENCH, surpassing the strongest baseline by 4.86 points and the no-memory variant by 2.08 points; its frozen memory transfers to other generators and improves WebGen-Bench performance from 54.56 to 57.94. Qualitative examples show clearer task structure and improved responsive layouts. Together, these results support reusable repair memory as a way to improve GenUI experiences across tasks.

1 INTRODUCTION

Generative user interfaces promise to turn a user's intent directly into a usable application. Language and vision-language models can reconstruct webpages, generate interactive websites, and create interfaces for individual requests (Si et al., 2025; Lu et al., 2026b; Leviathan et al., 2026). Realizing this promise requires coordinating appearance, behavior, and state: controls must be usable, actions must produce the intended outcomes, and user choices must persist through interaction. The central challenge is to make these requirements hold together as a complete experience.

Figure 1 (A) illustrates how this challenge becomes a burden on the user. A request for a studio-booking website produces overlapping time controls. After the user requests a layout correction, the revised interface still fails to respond to the booking action, prompting another round of feedback. Each round consumes additional generation tokens and asks the user to diagnose another defect. If the resulting repairs are discarded, a later request can encounter the same failures again. This motivates our central question: *how can a verified repair become reusable knowledge that helps a generator avoid repeating the same mistake?*

Feedback-driven generation and experience memory provide foundations for this goal. UICoder and WebGen-Agent use feedback to improve generated interfaces, while Reflexion and Voyager retain experience for subsequent decisions (Wu et al., 2024; Lu et al., 2026a; Shinn et al., 2023; Wang et al., 2023). For GUI generation, connecting repair to memory requires evidence that a change resolves its target failure while preserving the rest of the task. A screenshot can show that controls no longer overlap; source changes identify the intervention; interaction traces establish whether

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

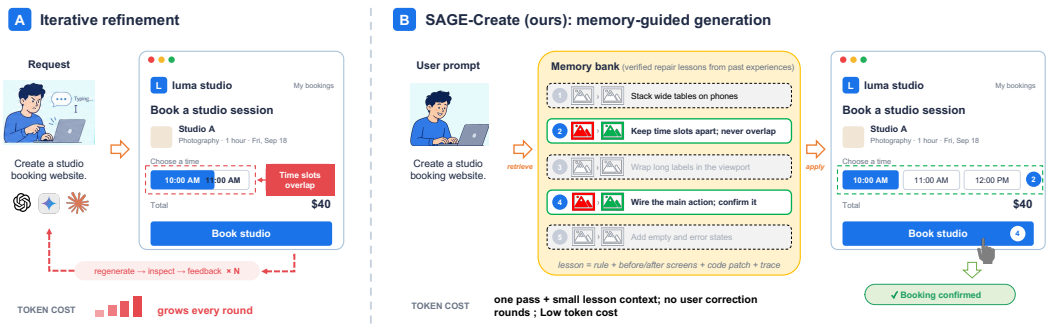


Figure 1: **From repeated user correction to memory-guided GUI generation. (A)** Iterative feedback exposes overlapping controls and an unresponsive booking action in successive rounds, each requiring another revision. **(B)** SAGE-CREATE retrieves verified repair lessons linking screenshots, code, and interaction traces to guide generation, then checks control layout, booking behavior, and preservation of user selections. Memory is intended to reduce repeated correction while retaining verification and refinement when needed.

booking succeeds and retains the selected studio, date, and time. Together, these modalities make a repair lesson inspectable and define the conditions under which it can be reused.

We introduce SAGE-CREATE, the generation method in SAGE (Self Adaptive Generative-UI Experiences), built around a *verified repair episode as both the unit of improvement and the unit of memory*. As Figure 1 (B) shows, lessons from earlier repairs guide generation before the user encounters the same defect. In our Reflect–Refine–Remember loop, Reflect makes task requirements explicit through an experience contract and retrieves applicable lessons. Refine generates and executes candidates, testing focused, reversible repairs through paired before-and-after replay. Remember admits a repair to multimodal memory only when the executed checks confirm both correction and preservation of required behavior. A learning curriculum adjusts lesson influence using evidence from distinct tasks. Adaptation therefore occurs through accumulated repair knowledge with fixed generator weights; every reused repair remains subject to verification on the current task.

Evaluating this approach requires testing the complete experience that a user can execute. We introduce SAGE-BENCH, comprising 300 prompt-only tasks across 11 experience families and 48 subfamilies. Its contracts specify required components, states, views, and interactions while allowing multiple valid designs. Our independent suite, SAGE-EVAL, combines source, browser, and perceptual evidence, with score ceilings for failures such as broken actions or severe overflow. Each held-out task starts from the same development memory checkpoint, and final evaluation remains outside the learning loop, allowing us to assess whether prior repair knowledge transfers to new tasks.

Our contributions are as follows:

- **Generation through verified repair.** SAGE-CREATE integrates experience contracts, candidate generation, runtime verification, and reversible repair to support complete user tasks.
- **Reusable multimodal memory.** Repair rules are linked to source changes, paired renders, and interaction traces. A curriculum governs their exploration, consolidation, and retention using verified support across tasks, enabling adaptation without updating generator weights.
- **Experience-level benchmark.** SAGE-BENCH provides 300 tasks across 11 families and 48 subfamilies, enabling comparison of diverse interface designs against explicit, shared requirements.
- **Independent experience evaluation.** SAGE-EVAL combines contract fidelity and execution reliability with perceptual alignment, visual layout, and data integrity, grounding scores in complementary evidence and explicit failure ceilings.

2 Related Work

GUI generation and verified repair. Generative UI creates interfaces for individual requests, and adaptive generative systems extend this idea to interfaces that change during use (Leviathan et al., 2026; Arora et al., 2026). UICoder uses compiler and multimodal feedback to filter training examples; WebGen-Agent combines screenshot and GUI-agent feedback with refinement and reinforcement learning; WebGrader develops executable flow contracts and reusable verifier skills (Wu et al., 2024; Lu et al., 2026a; Chen et al., 2026). Behavioral specifications and test-based repair have established foundations in Statecharts and GenProg (Harel, 1987; Le Goues et al., 2011). SAGE-CREATE connects these ideas through the repair episode: the same paired evidence that accepts a source change also determines whether it can become a positive memory lesson.

Self-improving agents and memory. Self-Refine revises outputs using feedback, Reflexion stores verbal experience, and Voyager maintains executable skills (Madaan et al., 2023; Shinn et al., 2023; Wang et al., 2023). Automated Design of Agentic Systems and the Darwin Gödel Machine search over agent implementations (Hu et al., 2025; Zhang et al., 2025b). For GUI operation, Darwinian Memory selects reusable trajectory units, while HyMEM combines symbolic knowledge and trajectory embeddings (Mi et al., 2026; Zhu et al., 2026). Our target is the generated interface itself. A lesson specifies a source-level repair with applicability conditions and visual, program, and behavioral evidence. Support from distinct tasks governs its influence, and retrieval may abstain when support is insufficient, following the broader principle of selective prediction (Geifman & El-Yaniv, 2019).

Benchmarks and experience evaluation. Design2Code studies screenshot reconstruction; Interaction2Code adds interactive behavior; and WebGen-Bench tests instruction-conditioned website functionality (Si et al., 2025; Xiao et al., 2024; Lu et al., 2026b). DesignBench and WebCompass cover generation, editing, and repair, while Vision2Web includes full-stack development (Xiao et al., 2025b; Lei et al., 2026; He et al., 2026). ArtifactsBench combines code and temporal screenshots, UI-Bench uses expert visual comparisons, and WebArena and VisualWebArena evaluate interaction with existing websites (Zhang et al., 2025a; Jung et al., 2025; Zhou et al., 2024; Koh et al., 2024). SAGE-BENCH complements these settings with a taxonomy organized by user intent and contracts spanning components, states, views, and actions. SAGE-EVAL assesses fulfillment of those contracts using complementary evidence and explicit score ceilings. Its proposed experience-quality formulations require separate human validation; they do not imply a validated measure of long-term user satisfaction. Appendix C.4 (Table C.4) compares nine GUI-generation benchmarks across task design, scale, human involvement, and evaluation scope. It positions our joint specification of states, views, actions, and data continuity, while distinguishing protocol coverage from demonstrated performance.

3 SAGE-Create: Learning from Verified Interface Repairs

We introduce SAGE-CREATE, a training-free agentic system that improves GUI generation by learning from its own repairs. Given a task, SAGE-CREATE generates candidate interfaces, executes them in a browser, and repairs the failures it observes. A repair is stored as a reusable *lesson* only if paired replay confirms that it resolves the target failure while preserving the behavior the task requires. Subsequent tasks retrieve these lessons before generation, which reduces the recurrence of previously repaired failures. Formally, for task l with input $u_l = (p_l, D_l, I_l, \mathcal{R}_l)$, comprising a prompt, optional source data, optional visual references, and structured requirements, the controller $\mathcal{A}_\theta$ produces an interface and updates its memory:

$$(G_l,\, M_{l+1},\, W_{l+1}) = \mathcal{A}_\theta(u_l,\, M_l,\, W_l;\, \mathcal{B}), \tag{1}$$

where G_l is the final GUI, M_l stores verified lessons, W_l stores warnings from rejected repairs, and $\mathcal{B}$ is the compute budget. Adaptation occurs only through memory; the generator parameters θ remain fixed. Figure 2 provides an overview.

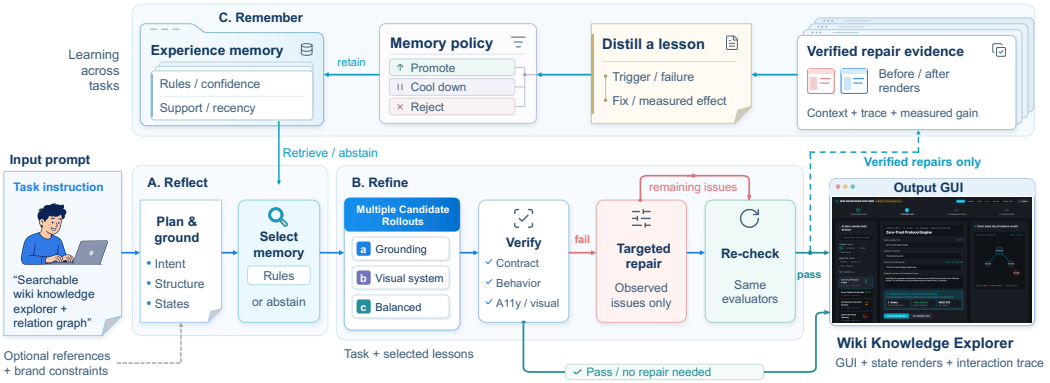


Figure 2: **Overview of SAGE-CREATE.** *Reflect* infers a task specification and retrieves relevant lessons. *Refine* generates candidates, runs them in a browser, and tests focused repairs with before-and-after replay. *Remember* stores accepted repairs as multimodal lessons (rule, code change, screenshots, interaction traces), and a curriculum decides which lessons are allowed to influence later tasks. Generator weights stay fixed, and benchmark scores never enter the loop.

3.1 REFLECT–REFINE–REMEMBER LOOP

Reflect. Given the prompt, SAGE-CREATE first infers a *task specification*

$$\mathcal{C}_l = \langle c_l,\, K_l,\, S_l,\, V_l,\, T_l \rangle, \tag{2}$$

where c_l is the task category from our benchmark taxonomy, K_l the component checklist, S_l the required states (e.g., loading, empty, error), V_l the required views (e.g., phone, dark mode), and T_l the interaction tasks with their expected outcomes. $\mathcal{C}_l$ serves three roles: it is the checklist the verifier tests during Refine, the metadata stored with every lesson learned on this task, and the retrieval query. Lessons are retrieved by matching the predicted category c_l and specification fields against those stored with each lesson, so only lessons learned in similar task contexts are reused.

Refine. The generator produces N candidate interfaces conditioned on $\mathcal{C}_l$ and the retrieved lessons. The verifier executes each candidate in a browser and checks every item of $\mathcal{C}_l$ (required components are present and usable, states are reachable, views render without overlap or clipping, and actions produce their expected outcomes), recording screenshots, action traces, and runtime logs. For each observed failure, the controller applies a small, reversible patch and replays the same checks before and after the edit. The patch is accepted only if the failure is resolved and all previously passing checks still pass; otherwise, it is rolled back and recorded as a warning. Finally, candidates with a blocking failure are discarded, and the rest are ranked by task completion, accessibility, and presentation to select the output.

Remember. Each accepted repair, together with its before-and-after evidence and the task specification $\mathcal{C}_l$, is stored in memory as a candidate lesson (Section 3.2).

3.2 MULTIMODAL MEMORY LEARNING CURRICULUM

Lessons. GUI failures are often visual and behavioral: a text rule alone cannot show *what* went wrong or confirm that a fix preserved the interface. Each lesson therefore stores one verified repair with multimodal evidence,

$$m = \langle x,\, f,\, r,\, E^-,\, E^+,\, \pi,\, h \rangle, \tag{3}$$

where $x = \mathcal{C}_l$ is the task specification, f the failure, r a reusable rule in natural language, E^-, E^+ the before-and-after screenshots and interaction traces, π the code patch, and h its reuse history. Figure 3 shows an example: at iteration 10, the generator produced an audio

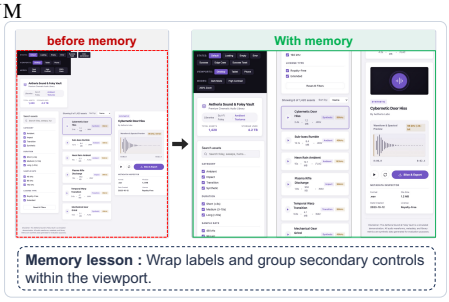


Figure 3: **A multimodal memory lesson.** Before/after renders are stored with the rule and reused on later tasks.

Table 1: **SAGE-BENCH taxonomy.** 11 experience families and 48 subfamilies; parentheses give the number of tasks (one per leaf pattern).

Family	Subfamilies (tasks)	Total
Information	Document Consumption (7); Search & Discovery (7); Monitoring (7); Reporting (7); Knowledge Exploration (7)	35
Operational	Transaction Processing (7); Workflow Execution (7); Case Management (7); Administration (7); Safety, Security & Compliance (6)	34
Decision	Analysis (7); Comparison (7); Planning (7); Forecasting (6); Recommendation (6); Optimization (6)	39
Creation	Content (7); Data (7); Asset (7); Configuration (7)	28
Collaboration	Communication (7); Review (7); Coordination (7); Shared Workspace (7)	28
Learning & Assessment	Learning Content (6); Assessment (5); Practice & Simulation (5); Curriculum & Progress (5)	21
Presentation & Narrative	Slide Decks (5); Storytelling Layouts (5); Carousel & Card Narrative (4); Speaker & Audience Support (5)	19
Commerce & Growth	Merchandising (5); Conversion (6); Campaigns (6); Customer Lifecycle (5)	22
Data & Scientific Comm.	Visualization (6); Scientific Figure (5); Data Exploration (5); Explainability (5)	21
Agent	Agent Assistant (7); Agent Workspace (7); Multi-Agent Workspace (7); Agent Operations Center (8)	29
System & Device	Device UI (7); Settings & Control (6); Installation & Setup (5); Environment-Specific UI (6)	24
11 families	**48 subfamilies**	**300**

library whose labels and secondary controls overflowed the viewport (left); the verifier flagged the failure, and the repair *"wrap labels and group secondary controls within the viewport"* passed replay (right), so the paired renders, patch, and rule were stored as a lesson. Retrieval matches the current specification (and, during repair, the observed failure) against stored specifications and rules, and the generator receives each retrieved rule with its patch and evidence excerpts; lessons whose context does not match are not retrieved.

Curriculum. Storing every accepted repair would fill the memory bank with redundant lessons and make retrieval less effective. The curriculum therefore governs how lessons are created, promoted, and removed, keeping the bank compact and effective: **Explore (creation):** an accepted repair that follows an existing lesson's rule strengthens that lesson instead of creating a new one; only a repair with a genuinely new rule enters the bank as a low-influence *candidate*. **Consolidate (promotion):** a candidate becomes *active* once it is supported by accepted repairs on several distinct tasks with a high success rate, unless a warning contradicts it. **Retain (pruning):** unused lessons are *cooled*, repeatedly failing lessons are *restricted*, and when the bank reaches its fixed capacity, the lowest-ranked inactive lesson is evicted.

As a result, the bank grows with new failure types rather than with repeated fixes, and only well-supported lessons remain active for retrieval. All thresholds and the capacity are fixed in advance. Memory is built only on development tasks, and every evaluation task starts from the same frozen memory.

4 SAGE-BENCH: A TAXONOMY OF GUI EXPERIENCES

Existing GUI-generation benchmarks group tasks by input type (e.g., screenshot or instruction) or by website category, which leaves the space of user experiences poorly covered. We instead organize tasks by *what the user is trying to accomplish*. SAGE-BENCH contains 300 tasks spanning 11 experience families and 48 subfamilies. Each task requires a self-contained, executable web interface and comes with a task specification (Eq. 2) that lists the required components, states, views, and interactions.

SAGE-BENCH Taxonomy. We propose a hierarchical taxonomy that organizes GUI generation tasks by user intent rather than by input modality or website genre. The hierarchy has three levels. *Experience families* capture the user's high-level goal, such as making a decision or collaborating with others; *subfamilies* capture the workflow through which that goal is pursued, such as comparison or review; and *leaves* capture concrete interface patterns that realize the workflow, such as a KPI dashboard or a code-review workspace. The design is grounded in task-abstraction frameworks, which separate a task's purpose from its inputs and interactions (Brehmer & Munzner, 2013), and in human–AI interaction guidelines that emphasize user control, correction, and feedback (Amershi et al., 2019). The resulting hierarchy comprises 11 families, 48 subfamilies, and 300 leaves, as shown inTable 1. It spans conventional web applications (information, operations, commerce) as well as experiences that existing benchmarks rarely cover, including collaboration, learning and assessment, scientific communication, agent supervision, and device control. Each leaf contributes

Table 2: **SAGE-BENCH results across 11 experience families.** n: number of tasks per family. Claude, Gemini Pro, and Gemini Flash denote Claude Opus 4.8, Gemini 3.1 Pro, and Gemini 3.5 Flash. SAGE-CREATE uses multimodal repair memory with the learning curriculum. Scores range from 0 to 100 (higher is better);

Experience family	n	Claude-4.8 -Sonnet	GPT-5	Gemini-3.1 Pro	Gemini-3.5 Flash	WebGen-Agent	SAGE-CREATE (Ours)
Information	35	64.18	66.86	65.04	65.84	62.70	**74.51**
Operational	34	63.54	66.94	65.90	65.62	62.00	**73.86**
Decision	39	65.59	67.80	65.44	66.28	63.90	**75.23**
Creation	28	66.34	67.72	66.04	65.13	63.60	**73.70**
Collaboration	28	66.68	69.12	67.86	66.66	65.24	**72.99**
Learning & Assessment	21	69.32	71.32	68.47	70.95	67.08	**76.79**
Presentation & Narrative	19	67.47	70.50	68.35	69.66	66.18	**73.02**
Commerce & Growth	22	67.53	70.12	68.57	68.10	66.12	**72.79**
Data & Scientific Comm.	21	67.78	70.44	68.10	69.39	66.32	**73.78**
Agent	29	67.49	69.50	66.85	68.12	65.52	**74.02**
System & Device	24	69.82	72.02	69.48	70.66	68.00	**74.06**
Overall	300	66.50	69.26	67.00	67.50	64.84	**74.12**

exactly one task, so family sizes (19–39 tasks) reflect the number of distinct interface patterns within each family rather than balanced sampling. We therefore report both task-weighted and family-level scores. Appendix C details how the benchmark tasks and their specifications are constructed, and Appendix C.4 compares SAGE-BENCH with existing GUI-generation benchmarks.

5 SAGE-EVAL: MEASURING GUI EXPERIENCE QUALITY

SAGE-BENCH specifies the intended experience; SAGE-EVAL measures how well a generated interface fulfills its task specification. SAGE-EVAL scores eleven dimensions on a 0–100 scale, grouped into three categories. The first two, *specification fidelity* and *execution reliability*, adapt established metrics from prior GUI-generation benchmarks. The third, *experience quality*, consists of metrics we propose. SAGE-EVAL scores only final outputs and is separate from the verifier used by SAGE-CREATE.

Specification fidelity comprises prompt alignment, component coverage, state coverage, view coverage, and task support. Following instruction-alignment judging and executable functional testing in prior benchmarks (Si et al., 2025; Xiao et al., 2024; Zhang et al., 2025a; Lu et al., 2026b), each coverage score is the percentage of required items in the task specification with observed passing evidence. A component must be usable, a state must be reached, a view must remain usable, and an action must produce its expected outcome. **Execution reliability** comprises build and render, accessibility, and safety/licensing/provenance, adapted from runtime and accessibility audits in prior work (Lu et al., 2026b; Liu et al., 2026); accessibility probes test selected WCAG 2.2 requirements rather than full conformance (W3C Accessibility Guidelines Working Group, 2023). These adopted dimensions check whether required elements exist and work, but not whether the experience as a whole is coherent, legible, and correct. We therefore propose three **experience-quality** metrics. *Perceptual alignment* measures whether the visible user journey communicates the requested task coherently. A video-capable model first describes sessions recorded on desktop, phone, and dark mode (content, hierarchy, feedback, transitions, and visible defects), and a second call scores this description against the task specification. Because visual polish cannot show that a control works, the score is capped whenever execution reveals failed controls, missing states, or rendering defects. *Visual layout* measures whether required content and controls remain legible and usable in every required view. It averages per-view scores from weighted rubric checks, with hierarchy, grouping, readability, and control visibility judged from screenshots and overlap, clipping, and overflow computed from element geometry; a missing render scores zero. *Data integrity* measures whether values and relationships remain correct through interaction and recovery (e.g., a checkout total must still match item quantities after an edit and after an error-and-retry). It scores data obligations, derived from the specification before execution, against evidence linking initial records to application state

and displayed values, and unverifiable obligations score zero. In the current prompt-only release, data integrity checks the internal consistency of synthetic data rather than external factual grounding.

Evidence and scoring. Each interface is evaluated from three complementary evidence sources: static source analysis, instrumented browser execution, and recorded interaction sessions. Scores are based only on runtime-verified behavior; static evidence such as an event handler is insufficient. Violations of necessary conditions (e.g., startup failure or an unresponsive required action) impose hard score ceilings on the affected dimensions. The aggregate score is the unweighted mean over applicable dimensions and is reported together with all per-dimension scores, so blocking failures remain visible. Scores reflect observed task support under fixed judges, rubrics, and execution settings, not long-term user satisfaction. Appendix D details metric definitions, evidence, ceilings, and judge configuration; Appendix C.4 compares SAGE-EVAL with existing protocols.

6 EXPERIMENTS

We evaluate SAGE-CREATE along four axes: end-to-end GUI quality against frontier models, the individual contributions of verified memory and the learning curriculum, transfer of lessons to other generators, and generalization to an external benchmark. All comparisons are run under matched conditions, so each gain can be attributed to a single component. Implementation details and additional results are provided in Appendix.

Experimental Setup. On SAGE-BENCH, SAGE-CREATE uses Gemini-3.5-Flash (temperature 0.3) with three initial candidates, at most one repair, and up to five retrieved lessons per task. Memory is built through 100 self-play iterations on development tasks, split into Explore (iterations 1–20), Consolidate (21–70), and Retain (71–100) stages; a lesson is promoted after three verified successes on distinct tasks, and the bank is capped at 256 lessons. Every held-out task then starts from the same frozen memory checkpoint, preventing any cross-task leakage. Each comparison fixes the generator, task specification, execution contexts, and budget, varying only the factor under study. SAGE-EVAL scores frozen outputs on a 0–100 scale; we report the task-weighted mean over 300 tasks alongside per-family scores. Full settings and ablation controls are given in Appendix.

6.1 MAIN RESULTS AND MEMORY ABLATION

SAGE-CREATE outperforms frontier models across all experience families. Table 2 compares SAGE-CREATE with frontier models and open-source WebGen-Agent Lu et al. (2026a) on all 300 SAGE-BENCH tasks. SAGE-CREATE reaches an overall score of 74.12, a 4.86-point improvement over the strongest baseline (GPT-5, 69.26), and leads in every one of the 11 experience families. The largest gains appear in Decision (+7.43) and Information (+7.65), which combine dense content with multiple interdependent states. Direct generation often breaks the consistency between views in these tasks, whereas specification-guided verification and repair preserve it.

Generator vs verified memory. Without memory, the SAGE-CREATE generator already scores 72.04 (Table 3), 4.54 points above direct generation, reflecting the combined effect of specification inference, candidate generation, verified repair, and selection. It also surpasses Self-Refine (69.2) and Best-of-3 sampling without verification (70.5), indicating that the gain comes from execution-grounded verification rather than additional sampling alone. Adding memory improves performance monotonically: text lessons contribute +0.44, multimodal evidence a further +0.53, and the learning curriculum +1.11, for a cumulative gain of 2.08 points. The curriculum yields the largest single gain, suggesting the value of governance in memory learning.

Memory vs. simpler alternatives. To test whether the gains come from learned experience rather than generic guidance or extra computation, we compare against matched controls (Table 4). The static-checklist control keeps the generator but replaces learned lessons with fixed design guidance. All controls share the generator, task inputs, four-candidate budget, external evaluator, and repair allowance, except the explicit no-repair Best-of-4 reference. Each primary condition covers all 300 tasks with three independent generation repetitions. The static checklist recovers only 0.57 of the 2.08-point memory gain; verified memory retains a 1.51-point advantage, with 4.5 points higher action completion and 4.3 points fewer blocking failures, at 2.2k additional tokens per task. The benefit therefore stems from task-specific repair experience rather than generic design advice. Further matched contrasts isolate each mechanism: the generator exceeds Best-of-4 with one repair

Table 5: **Transfer of SAGE-CREATE memory to proprietary models.** Each model generates interfaces directly in reasoning mode, without memory (No) or with context from the frozen SAGE-CREATE memory (+Mem). Δ denotes absolute score-point gain.

		Gemini-3.5-Flash			GPT-5			Claude-4.8-Opus		
Experience family	n	No	+Mem	Δ	No	+Mem	Δ	No	+Mem	Δ
Information	35	65.84	**67.98**	+2.14	66.86	**68.94**	+2.08	64.18	**66.07**	+1.89
Operational	34	65.62	**67.38**	+1.76	66.94	**68.53**	+1.59	63.54	**65.37**	+1.83
Decision	39	66.28	**68.71**	+2.43	67.80	**70.21**	+2.41	65.59	**67.40**	+1.81
Creation	28	65.13	**66.45**	+1.32	67.72	**69.03**	+1.31	66.34	**67.48**	+1.14
Collaboration	28	66.66	**67.75**	+1.09	69.12	**70.24**	+1.12	66.68	**67.96**	+1.28
Learning & Assessment	21	70.95	**72.62**	+1.67	71.32	**73.35**	+2.03	69.32	**71.25**	+1.93
Presentation & Narrative	19	69.66	**70.29**	+0.63	70.50	**71.08**	+0.58	67.47	**67.94**	+0.47
Commerce & Growth	22	68.10	**69.51**	+1.41	70.12	**71.58**	+1.46	67.53	**68.42**	+0.89
Data & Scientific Comm.	21	69.39	**71.31**	+1.92	70.44	**72.57**	+2.13	67.78	**69.76**	+1.98
Agent	29	68.12	**70.18**	+2.06	69.50	**71.37**	+1.87	67.49	**68.76**	+1.27
System & Device	24	70.66	**71.48**	+0.82	72.02	**72.94**	+0.92	69.82	**70.88**	+1.06
Overall	**300**	67.50	**69.15**	+1.65	68.98	**70.64**	+1.65	66.56	**68.03**	+1.47

Table 6: **Deployment cost per GUI.** R: generation rounds; N: candidates; Ours: SAGE-CREATE. Iterative refinement uses GPT-5 with reasoning enabled.

Method	Setting	Tok.(k)	Min.
Iter. refinement	$R{=}1$	12.0	–
	$R{=}3$	63.0	15
	$R{=}5$	114.0	$\approx$22
Ours, no verifier	$N{=}1$	13.1	–
	$N{=}2$	45.2	–
Ours, full	$N{=}3$	**95.5**	**6–8**

Table 7: **WebGen-Bench results.** WebGen: WebGen-LM-32B; Claude: Opus 4.8; Gemini: 3.5 Flash. The mean averages overlapping groups and is diagnostic, not overall accuracy.

				SAGE-CREATE		
Category	**WebGen**	**Claude**	**Gemini**	**No**	**+Mem**	**Δ**
Content Pres.	46.60	50.10	56.12	60.92	**64.18**	+3.26
User Interact.	33.20	47.00	51.26	54.95	**57.35**	+2.40
Functional	29.10	38.18	40.38	46.17	**48.66**	+2.49
Data Display	43.00	51.66	53.82	56.18	**61.56**	+5.38
Mean	37.98	46.74	50.40	54.56	**57.94**	+3.38

by 0.69 points, verified admission exceeds target-only admission by 1.20, and the curriculum adds 0.94 with the same lesson pool and retrieval count.

Memory transfers across models. Because lessons are stored as model-agnostic rules paired with code and multimodal evidence, the frozen SAGE-CREATE memory can be supplied as context to other generators without adaptation (Table 5). Memory improves all three models: Gemini-3.5-Flash from 67.50 to 69.15 (+1.65), GPT-5 from 68.98 to 70.64 (+1.65), and Claude-4.8-Opus from 66.56 to 68.03 (+1.47), with gains in every family. Gains are largest in Decision (up to +2.43) and smallest in Presentation & Narrative (+0.47 to +0.63). Macro-averaged gains (+1.57, +1.59, +1.41) closely match the task-weighted ones, indicating that the benefit is not driven by the largest families.

Cost and latency. Beyond quality, SAGE-CREATE is efficient to deploy (Table 6). Iterative refinement (GPT-5 with reasoning) pays for every fix with another round, whereas SAGE-CREATE front-loads verified lessons into generation. The full pipeline uses 95.5k tokens, 16% fewer than five-round refinement, and runs in 6–8 minutes, about 3$\times$ faster than five rounds and 2$\times$ faster than three. Without verification, memory adds only 1.1k tokens over a single refinement draft, and two candidates cost 45.2k, 60% less than five rounds; these reduced variants do not inherit the full-system quality scores.

Memory transfers to an external benchmark. To test generalization beyond SAGE-BENCH, we evaluate on WebGen-Bench (Lu et al., 2026b), which changes both the tasks and the evaluator: 101 website instructions are scored by 647 browser-executed tests whose cases and expected outcomes are withheld during generation. Reusing the frozen SAGE-CREATE memory without any adaptation improves all four test groups (Table 7), raising the selected-group mean from 54.56 to 57.94 (+3.38). The largest gain is in Data Display Testing (+5.38), consistent with the memory's emphasis on layout and value preservation. Because the groups overlap, their mean is a diagnostic summary rather than overall test accuracy.

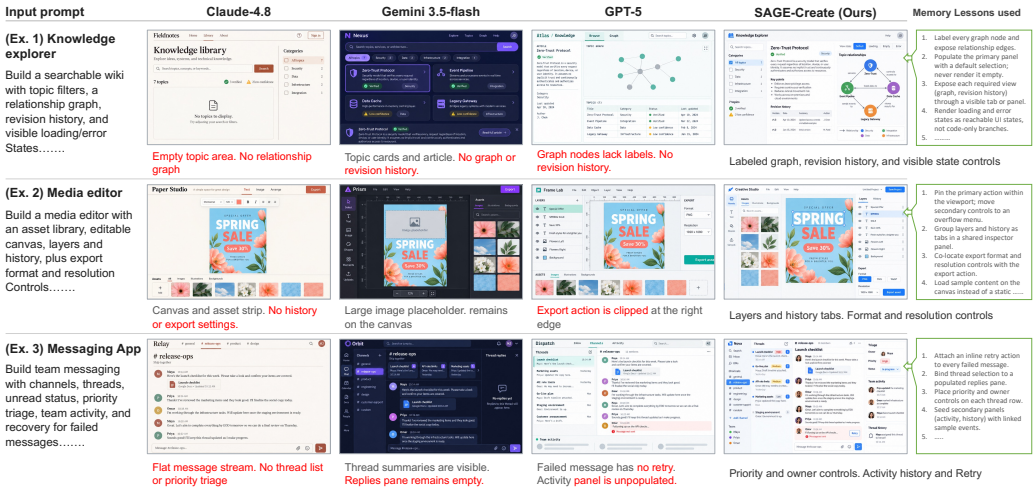


Figure 4: **Qualitative comparison on three SAGE-BENCH prompts.** Red text marks baseline limitations. With retrieved lessons (e.g., *label every graph node*, *pin the primary action within the viewport*, *attach an inline retry to failed messages*), SAGE-CREATE produces a labeled graph with revision history, an editor with layers, history, and export controls, and a triaged messaging workspace with retry. Screenshots show coverage; correctness is verified by execution.

6.2 QUALITATIVE ANALYSIS

Figure 4 compares proprietary models Claude-4.8, Gemini-3.5-Flash, GPT-5, and SAGE-CREATE on three shared prompts. These proprietary models often omit or hide required structure: the knowledge explorer shows an empty topic area, no relationship graph, or unlabeled nodes; the media editor lacks history and export settings, leaves a placeholder canvas, or clips the export action; and the messaging app collapses into a flat stream, leaves the replies pane empty, or offers no retry for failed messages. Guided by retrieved lessons such as *label every graph node*, *pin the primary action within the viewport*, and *attach an inline retry to failed messages*, SAGE-CREATE produces a labeled graph with revision history and state controls, an editor with layers, history, and export controls, and a messaging workspace with thread triage, activity history, and recovery. These cases illustrate task coverage and control visibility rather than benchmark-wide frequency; functional correctness is verified by execution. Appendix A.1 presents further examples of memory-guided repairs, pairing outputs without and with memory alongside the lesson applied. See Appendix A for more.

Table 3: **Memory ablation on SAGE-BENCH.** Top: generator variants; bottom: memory components added cumulatively to the same generator. $\Delta_{\text{prev}}/\Delta_0$: gain over the previous row / the no-memory generator.

Configuration	Score	Δ_{prev}	Δ_0
Gemini-3.5-Flash (direct)	67.50	–	–
Self-Refine (adapted)	69.2	–	–
SAGE-CREATE, direct generation	69.6	–	–
SAGE-CREATE, Best-of-3 (no verifier)	70.5	–	–
SAGE-CREATE w/o memory	72.04	–	0.00
+ text memory	72.48	+0.44	+0.44
+ multimodal evidence	73.01	+0.53	+0.97
+ curriculum (full)	**74.12**	+1.11	+2.08

Table 4: **Memory vs. static guidance.** 900 runs (300 tasks $\times$ 3) per condition. Action: mean fraction of required actions completed; Block.: runs with a critical failure; tokens cover generation, verification, selection, and repair.

Generator	Score	Action	Block.	Tok.(k)
No memory	72.04	76.4%	15.0%	89.0
+ static checklist	72.61	78.1%	13.0%	93.3
+ verified memory & curr.	**74.12**	**82.6%**	**8.7%**	95.5

7 CONCLUSION

We presented SAGE-CREATE, which enables GUI generators to improve across tasks by learning from verified multimodal repairs. Task specifications define what an experience requires, paired replay admits only repairs that preserve required behavior, and a curriculum governs which lessons are reused. On SAGE-BENCH, evaluated with SAGE-EVAL, SAGE-CREATE surpasses the strongest baseline by 4.86 points, with memory and curriculum contributing 2.08; the frozen memory also transfers to other models and to WebGen-Bench. Future work will assess, through user studies, how these gains translate into sustained usability and trust.

USE OF AI

AI tools were used to assist with grammar correction, checking formatting consistency, and verifying the consistency of figures with their captions and the manuscript text. All AI-assisted suggestions were reviewed by the authors, who retain full responsibility for the scientific content, accuracy, and integrity of the paper.

ETHICS STATEMENT

This work studies the automatic generation of graphical user interfaces (GUI). SAGE-BENCH consists of synthetic, prompt-only tasks constructed from our experience taxonomy; it contains no personal data, and all records displayed in generated interfaces are synthetic. Generator and judge models are accessed through their official APIs in accordance with their terms of use. Because SAGE-EVAL relies partly on model-based judgments, its scores may inherit biases of the judge models; we mitigate this by pairing every perceptual judgment with execution-based checks and score ceilings. As with any interface-generation technology, the method could be misused to produce deceptive interfaces, such as phishing pages that imitate real services. Our tasks exclude such content, and we encourage responsible downstream use consistent with the ICLR Code of Ethics. Conversely, the explicit accessibility checks in our specifications and evaluation are intended to encourage generated interfaces that remain usable for people relying on keyboard navigation, text scaling, and high-contrast modes.

REPRODUCIBILITY STATEMENT

We describe the Reflect–Refine–Remember loop in Sec. 3 and the multimodal memory and learning curriculum in Sec. 3.2, with the full algorithm, lesson schema, retrieval procedure, and curriculum thresholds in Appendix B. The SAGE-BENCH taxonomy and task construction are described in Sec. 4 and Appendix C, and all SAGE-EVAL metrics, evidence sources, browser conditions, score ceilings, and judge settings are specified in Sec. 5 and Appendix D. Experimental settings, including model versions, decoding parameters, candidate and repair budgets, matched-control configurations, and token accounting, are reported in Sec. 6. Because proprietary model APIs can be nondeterministic, we fix model versions, prompts, and temperatures, and report results averaged over repeated runs.

REFERENCES

Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N Bennett, Kori Inkpen, et al. Guidelines for human-ai interaction. In *Proceedings of the 2019 chi conference on human factors in computing systems*, pp. 1–13, 2019.

Piyush Arora, Ram Vivekananda, Wendy Yun, Colby Hawker, and Jaime Sonoda. Self-evolving systems: Moving beyond deterministic interfaces to adaptive generative interfaces. Technical report, 2026.

Matthew Brehmer and Tamara Munzner. A multi-level typology of abstract visualization tasks. *IEEE transactions on visualization and computer graphics*, 19(12):2376–2385, 2013.

Boshui Chen, Huiping Liu, and Shaolei Zhang. Webgrader: Training llms for web development with self-evolving programmatic grader. *arXiv preprint arXiv:2608.06474*, 2026.

Yonatan Geifman and Ran El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *International conference on machine learning*, pp. 2151–2159. PMLR, 2019.

David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

Zehai He, Wenyi Hong, Zhen Yang, Ziyang Pan, Mingdao Liu, Xiaotao Gu, and Jie Tang. Vision2web: A hierarchical benchmark for visual website development with agent verification. *arXiv preprint arXiv:2603.26648*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Sam Jung, Agustin Garcinuno, and Spencer Mateega. Ui-bench: A benchmark for evaluating design capabilities of ai text-to-app tools. *arXiv preprint arXiv:2508.20410*, 2025.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. Genprog: A generic method for automatic software repair. *Ieee transactions on software engineering*, 38(1):54–72, 2011.

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

Yaniv Leviathan, Dani Valevski, Matan Kalman, Danny Lumen, Eyal Segalis, Eyal Molad, Shlomi Pasternak, Vishnu Natchu, Valerie Nygaard, James Manyika, et al. Generative ui: Llms are effective ui generators. *arXiv preprint arXiv:2604.09577*, 2026.

Chenxu Liu, Yingjie Fu, Wei Yang, Ying Zhang, and Tao Xie. Webcoderbench: Benchmarking web application generation with comprehensive and interpretable evaluation metrics. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11632–11666, 2026.

Zimu Lu, Houxing Ren, Yunqiao Yang, Ke Wang, Zhuofan Zong, Junting Pan, Mingjie Zhan, and Hongsheng Li. WebGen-Agent: Enhancing interactive website generation with multi-level feedback and step-level reinforcement learning. In *International Conference on Learning Representations*, 2026a. URL https://arxiv.org/abs/2509.22644.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch. *Advances in Neural Information Processing Systems*, 38, 2026b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Hongze Mi, Yibo Feng, WenJie Lu, Song Cao, Jinyuan Li, Yanming Li, Xuelin Zhang, Haotian Luo, Songyang Peng, He Cui, et al. Darwinian memory: A training-free self-regulating memory system for gui agent evolution. *arXiv preprint arXiv:2601.22528*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2Code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, 2025. URL https://arxiv.org/abs/2403.03163.

W3C Accessibility Guidelines Working Group. Web content accessibility guidelines (WCAG) 2.2. W3C Recommendation, 2023. URL https://www.w3.org/TR/WCAG22/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Jason Wu, Eldon Schoop, Alan Leung, Titus Barik, Jeffrey P. Bigham, and Jeffrey Nichols. UICoder: Finetuning large language models to generate user interface code through automated feedback. *arXiv preprint arXiv:2406.07739*, 2024. URL https://arxiv.org/abs/2406.07739.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zhiyao Xu, and Michael R Lyu. Interaction2code: How far are we from automatic interactive webpage generation. *arXiv preprint arXiv:2411.03292*, 1, 2024.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu, Yuhang Wang, and Michael R Lyu. Interaction2code: Benchmarking mllm-based interactive webpage code generation from interactive prototyping. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 241–253. IEEE, 2025a.

Jingyu Xiao, Ming Wang, Man Ho Lam, Yuxuan Wan, Junliang Liu, Yintong Huo, and Michael R Lyu. Designbench: A comprehensive benchmark for mllm-based front-end code generation. *arXiv preprint arXiv:2506.06251*, 2025b.

Chenchen Zhang, Yuhang Li, Can Xu, Jiaheng Liu, Ao Liu, Changzhi Zhou, Ken Deng, Dengpeng Wu, Guanhua Huang, Kejiao Li, et al. Artifactsbench: Bridging the visual-interactive gap in llm code generation evaluation. *arXiv preprint arXiv:2507.04952*, 2025a.

J Zhang, S Hu, C Lu, R Lange, and J Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. arxiv [preprint](2025), 2025b.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Sibo Zhu, Wenyi Wu, Kun Zhou, Stephen Wang, and Biwei Huang. Hybrid self-evolving structured memory for gui agents. *arXiv preprint arXiv:2603.10291*, 2026.

APPENDIX

The appendix opens with qualitative results, then follows the order of the paper: the generation method, the benchmark, the evaluation suite, and the experiments.

- Appendix A: memory-guided repairs, eight further paired comparisons with the rules they retrieved, and the behavior each case must preserve.
- Appendix B: the complete Reflect–Refine–Remember procedure, the repair acceptance rule, the lesson schema and curriculum, a worked repair manifest, and the implementation and checkpoint profile.
- Appendix C: the full SAGE-BENCH taxonomy, task construction, and a comparison with nine existing benchmarks.
- Appendix D: SAGE-EVAL metric definitions, evidence, score ceilings, and aggregation.
- Appendix E: validation of SAGE-EVAL against expert ratings and injected defects.

A QUALITATIVE RESULTS

Open-ended GenUI experiences. Figures .5–.8 show SAGE-CREATE outputs for four open-ended creative prompts outside SAGE-BENCH. Each prompt asks for a multi-section experience with a distinct visual identity rather than a single page. For each, we show screenshots of the major sections of one generated website. Across an editorial architecture studio, a retro-futuristic operating system, a cinematic travel guide, and a space-exploration portal, the generated interfaces follow the requested aesthetic consistently across sections while keeping navigation, interactive components, and content structure intact. Figure .8 also shows a remaining failure mode (red dashed boxes). In the spacecraft technical explorer, selecting the James Webb Space Telescope displays an image of Mars, and selecting the Orion crew spacecraft displays a galaxy. These images fit the overall theme of the website but not the item they illustrate. Our verifier checks that images load and fit their layout but not that they match the semantics of their section, so this mismatch is not currently detected. This points to a need for content-level grounding checks in future work.

Figure 4 in the main paper compared SAGE-CREATE with frontier models on shared prompts. Here we hold the generator fixed and switch memory on and off, so that differences between paired outputs come from the retrieved lessons. Section A.1 walks through four repairs in detail, Section A.2 adds eight tasks from other families, and Section A.3 lists the behavior each case must preserve for the visible change to count. The cases were chosen to illustrate how lessons are reused; they are not a frequency estimate over the benchmark.

A.1 MEMORY-GUIDED REPAIRS IN ACTION

Figure A.1 shows four tasks, each generated by SAGE-CREATE without and with memory, next to the lesson that was retrieved.

Hierarchy. In panel (a), an infrastructure configurator moves the total ahead of the repeated line items, so the value the user is deciding on is seen first. In panel (b), a policy interface makes readiness the focal summary and keeps the supporting citations in a separate evidence rail. Both lessons concern priority and grouping, which the visual-layout metric scores; whether the total and the citation links are still correct is checked separately by the data-integrity probes.

Responsive layout. In panel (c), an audio library wraps long labels and groups its secondary controls so that every action stays inside the viewport; this is the lesson shown in Figure 3. In panel (d), a version-history table becomes a list of labeled records on narrow screens, keeping every field and the restore action. This is the running example of Section 3.2. Fitting the content on screen is the visible part of the fix. Record identity and restoring the selected version are behavioral obligations that paired replay has to confirm, and a patch that simply hid the overflowing column would be rejected because it hides a required action.

Input Prompt: Create an editorial-style website for a high-end architecture studio named \u201cMONOLITH.\u201d Use oversized typography, dramatic full-screen architectural photography, asymmetric layouts, generous whitespace, project case studies, an interactive project gallery, subtle image-reveal animations, and a minimal navigation system. The overall experience should feel like an award-winning architecture magazine rather than a conventional business website.

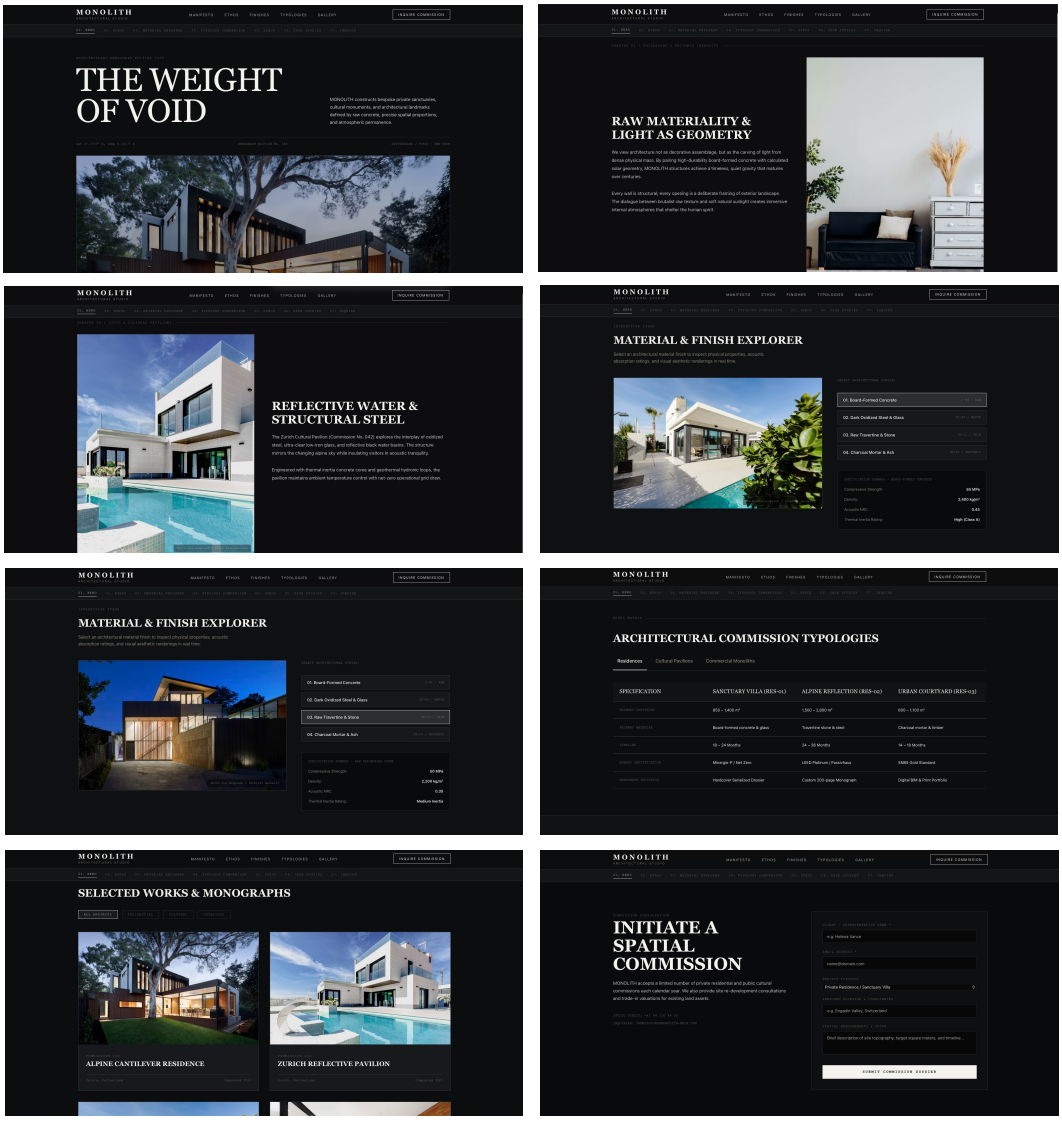


Figure .5: **Editorial architecture studio ("MONOLITH").** SAGE-CREATE produces an editorial-style website with oversized typography, full-bleed architectural photography, asymmetric layouts, a material and finish explorer, architectural typologies, selected works, and a commission form, maintaining a consistent magazine-like visual identity across sections.

A.2 ADDITIONAL MEMORY-GUIDED COMPARISONS

Figure A.6 covers eight more tasks: data annotation, analyst assistance, lesson delivery, a design workspace, poster/banner creation, subscription management, system monitoring, and a discussion forum. Each row gives the prompt, the output without memory, the retrieved rules, the output with memory, and the evaluation dimensions with the largest gains.

Input Prompt: Design a website that behaves like a fictional computer operating system from the year 2086 imagined by designers in 1986. Include movable windows, desktop icons, terminal commands, system notifications, music player, file explorer, artificial intelligence assistant, and hidden easter eggs. Combine retro CRT aesthetics, pixel typography, scanlines, futuristic diagrams, and satisfying UI sound-inspired visual feedback. Make every major element interactive.



Figure .6: **Retro-futuristic operating system ("2086 as imagined in 1986").** SAGE-CREATE generates a desktop-like experience with movable windows, terminal panels, system notifications, and multiple CRT-style color themes, combining pixel typography and scanline aesthetics with interactive window controls.

A few rules recur across these tasks: fluid sizing, avoiding horizontal overflow, and self-contained semantic HTML. Others are narrower, such as focus indicators that survive forced-color mode, inline error recovery, and usability at 200% text zoom. The resulting changes are mostly structural: canvas and inspector panels are rearranged, dashboards gain a summary row, and records are grouped. Scores improve in all eight pairs, spread over layout, accessibility, state and component coverage,

Input Prompt: Create a cinematic travel website titled \u201c72 Hours in Kyoto.\u201d Structure the experience around morning, afternoon, evening, and night across three days. Include immersive destination photography, animated maps, temples, hidden streets, restaurants, cultural experiences, itinerary cards, travel tips, and atmospheric transitions. Blend traditional Japanese visual influences with refined contemporary web design.

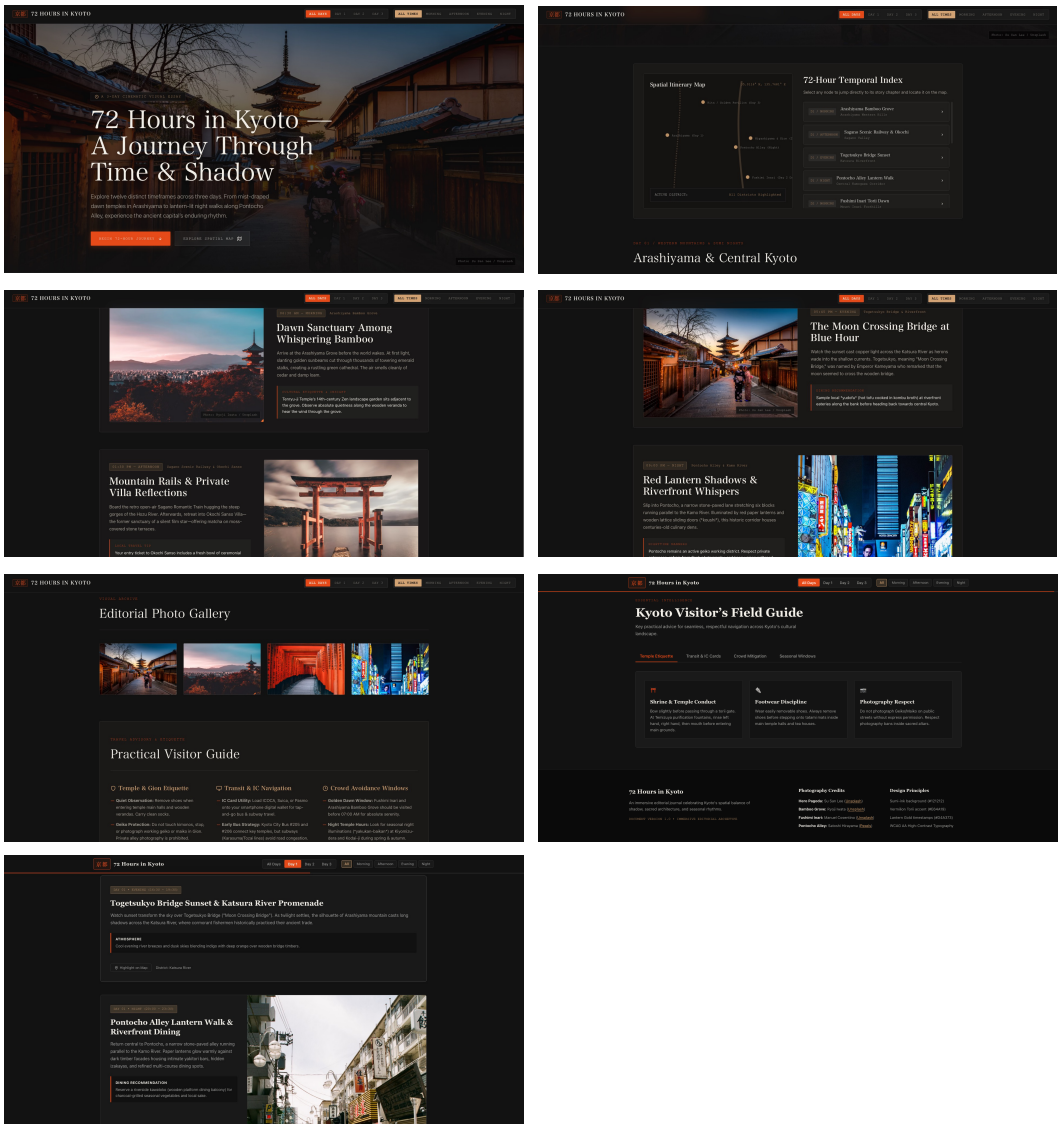


Figure .7: **Cinematic travel guide (“72 Hours in Kyoto”).** SAGE-CREATE structures the experience around times of day across three days, with an immersive hero, itinerary cards for temples, streets, and restaurants, a photo gallery, practical travel tips, and a visitor field guide, blending traditional Japanese influences with contemporary layout.

data integrity, and build checks. Several rules are retrieved together for each task, so these examples tie rules to changes without isolating the effect of any single rule.

Figures A.2–A.5 show individual memory lessons in detail. Each figure pairs one rule with its before-and-after renders, the measured effect of the repair, the CSS patch, and the task in which it was learned or reused. The four figures cover four failure categories. *Responsive structure* (Fig-

Input Prompt: Build an immersive space exploration website called \u201cORBIT.\u201d The hero section should show Earth floating in space with orbital paths, satellites, and mission markers. Include live-style mission statistics, upcoming launches, planet exploration cards, spacecraft details, a mission timeline, and an interactive star-map-inspired section. Use deep blacks, cosmic gradients, glowing UI elements, technical typography, and cinematic transitions.

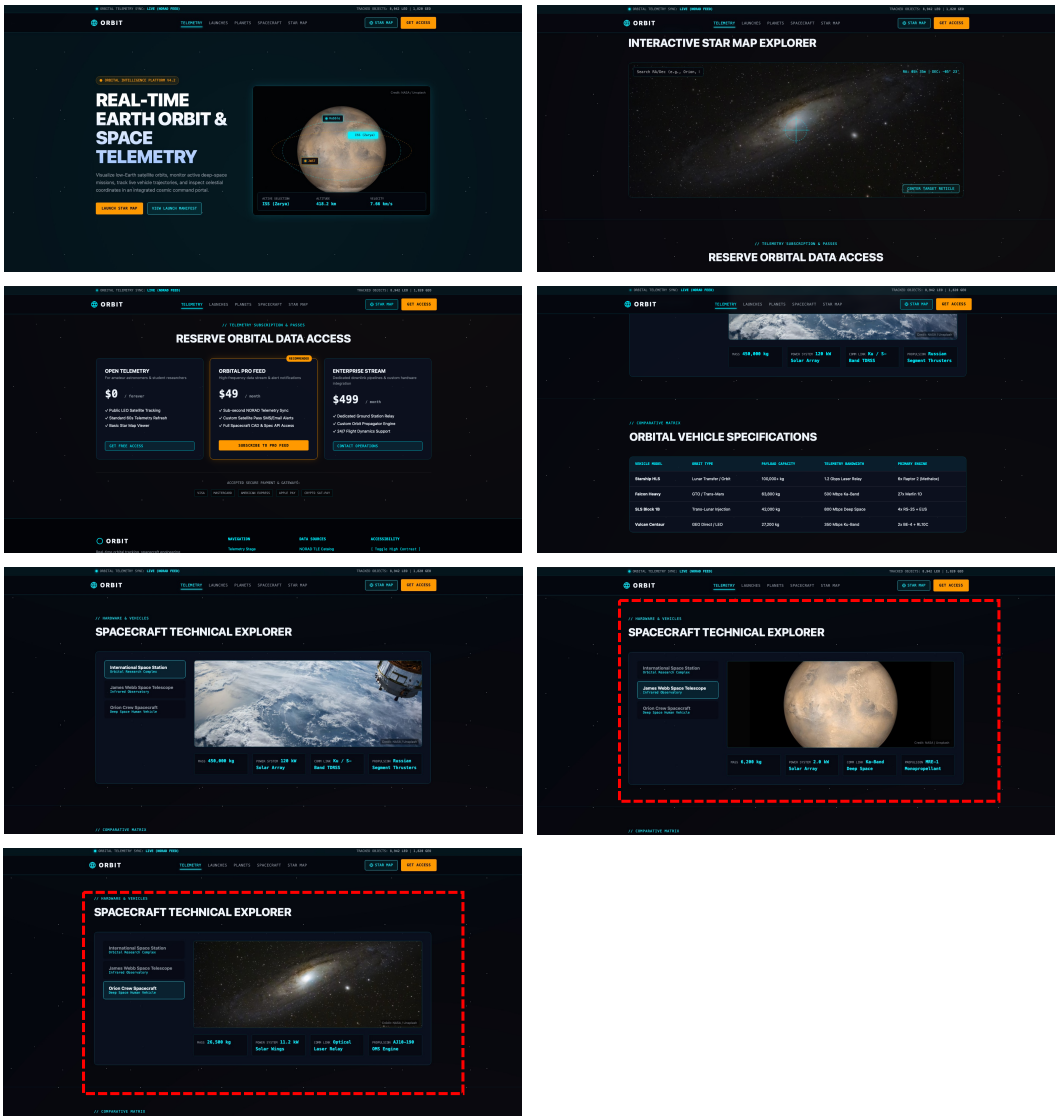


Figure .8: **Space-exploration portal ("ORBIT") with a failure case.** SAGE-CREATE generates a hero with Earth and orbital telemetry, mission data, vehicle specifications, a spacecraft technical explorer, and a star-map section in a consistent dark, glowing style. Red dashed boxes mark a remaining failure: in the spacecraft explorer, the images shown for the James Webb Space Telescope (Mars) and the Orion crew spacecraft (a galaxy) match the website's space theme but not the selected item. Our checks verify that images load and fit their layout, not that they match their section's content.

ure A.2): a retrieved intrinsic-sizing rule removes fixed viewport assumptions in a permit workflow, reducing overflow from 89 px to 0 and eliminating all overlaps and clipped elements on a phone viewport. *Layout boundaries* (Figure A.3): a newly created rule collapses a report's sidebars into a

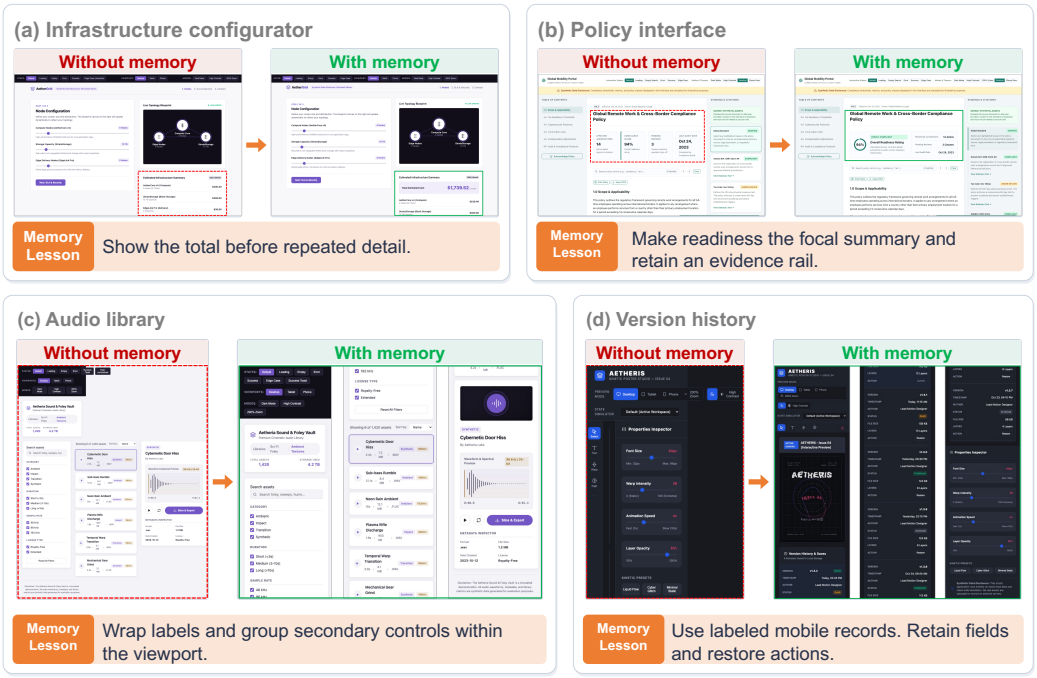


Figure A.1: **Memory in action.** Each panel pairs outputs without and with memory alongside the retrieved lesson. Red dashed boxes mark presentation issues; green boxes mark the revised regions. (a) Total placed before repeated details. (b) Readiness summary with a retained evidence rail. (c) Wrapped labels and grouped secondary controls within the viewport. (d) Labeled mobile records that retain fields and restore actions. Screenshots show layout changes; preservation of values, associations, and interactions is verified by execution.

single phone column, reducing overflow from 251 px to 0. *State feedback* (Figure A.4): an explicit loading state with visible progress and a recovery action is added to a speaker-notes workspace. *Hierarchy* (Figure A.5): repeated cards in a policy viewer are replaced by a focal summary and a quieter evidence rail, reducing visible cards from 23 to 7. The figures also report residual issues, such as one remaining overlap in the report, four remaining clipped elements in the workspace, and one additional overlap in the policy viewer. These show that a lesson fixes its target failure without guaranteeing that every other layout check improves.

A.3 What Each Case Must Preserve

A screenshot answers the visible question for each case, but not whether the workflow behind it still works. Table A.1 pairs the two for the eight tasks above. The right-hand column lists the checks a change must pass under the task contract; it is a specification, not an additional result.

Some rules also fix the mode in which they must be checked. A forced-color focus rule is only meaningful in high-contrast mode, a zoom rule at 200% text scaling, and an error-recovery rule along the failure path it targets. This is why a better-looking after-image is never enough to admit a lesson: positive admission requires the complete target and protected replay described in Appendix B.4.

Responsive structure and content

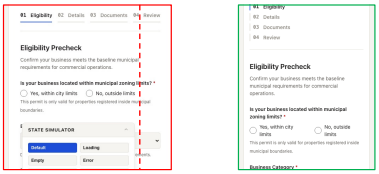
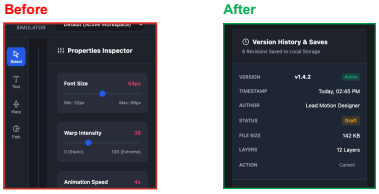
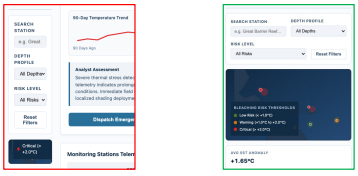
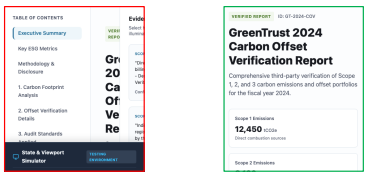
Memory element / rule	Visual evidence	Effect + source
M01 rule-000010	**Intrinsic sizing for a permit workflow**	
Remove fixed viewport assumptions and constrain media, grids, and controls with intrinsic widths and overflow-safe containers. **Example context** Long-form permit application	Before / After  Dashed: phone edge. The stepper becomes vertical and the form fits the viewport.	**Observed (before → after)** Overflow (px) **89 → 0** Overlap count **20 → 0** Clipped elements **6 → 0** **CSS excerpt** `.stepper flex-direction: column; .workspace-main min-width: 0;`
390 × 844 px · default	v4 · ep.07 · Retrieved rule	trial-b7ca1426b71f
M02 rule-000015	**Component breakpoints for supporting charts**	
Define component-level breakpoints that preserve the dominant task while rails, tools, and supporting content reflow below it. **Example context** Executive briefing / slide preview	Before / After  Corresponding chart regions at different page positions: a squeezed chart becomes a full-width module.	**Observed (before → after)** Overflow (px) **703 → 0** Clipped elements **5 → 0** **CSS excerpt** `.slide-grid-layout /* below 768 px */ grid-template-columns: 1fr !important;`
390 × 844 px · default	v3 · ep.43 · Stored repair evidence	trial-d1829df6cb93
M03 rule-000018	**Labeled mobile records for version history**	
Transform wide tables or timelines into an accessible mobile representation while retaining labels and relationships. **Example context** Poster studio / version history	Before / After  Different page regions: collapsed workspace (before); labeled version record (after).	**Observed (before → after)** Overflow (px) **214 → 0** Overlap count **6 → 0** Clipped elements **5 → 0** **CSS excerpt** `td::before { content: attr(data-label); }`
390 × 844 px · default	v3 · ep.53 · Stored repair evidence	trial-23c47fe1d5f7
M04 rule-000019	**Fluid sizing restores filters and map**	
Use fluid type, minmax grids, and intrinsic sizing so the composition adapts continuously rather than only at one fixed breakpoint. **Example context** Coral-bleaching risk monitor	Before / After  Corresponding filter/map regions at different page positions: a cramped rail becomes a usable full-width view.	**Observed (before → after)** Overflow (px) **179 → 0** Overlap count **4 → 1** *One overlap detection remains.* **CSS excerpt** `.main-layout grid-template-columns: minmax(0, 1.2fr) minmax(0, 1fr);`
390 × 844 px · default	v3 · ep.21 · Stored repair evidence	trial-43a0ecdee689

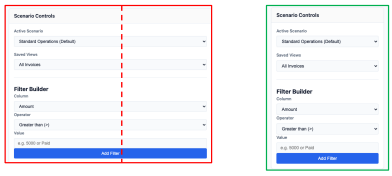
Figure A.2: **Memory lesson: responsive structure and content.** Rule M01 (retrieved), learned from a long-form permit application: remove fixed viewport assumptions and constrain media, grids, and controls with intrinsic widths and overflow-safe containers. On a 390×844 phone viewport, the stepper becomes vertical and the form fits the screen, reducing overflow from 89 px to 0, overlaps from 20 to 0, and clipped elements from 6 to 0. The panel shows the rule, before/after renders (dashed line: phone edge), measured effects, and the CSS patch.

Layout boundaries and readable content

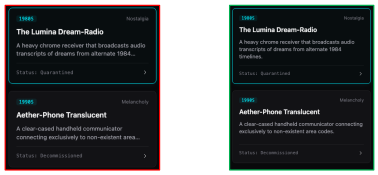
Memory element / rule	Visual evidence	Effect + source
M05 rule-000001	**A usable phone layout for a report**	

Implement the phone mode with observable responsive or accessibility behavior.
Example context
Carbon-offset verification report
	Before / After
	
	Matched phone-width detail: sidebars squeeze the report before; a single column restores the title and metrics.
		Observed (before → after)
		Overflow (px) **251 → 0**
		Overlap count **9 → 1**
		Clipped elements **3 → 0**
		One overlap detection remains.
		CSS excerpt
		`.workspace-grid`
		`/* below 768 px */`
		`grid-template-columns: 1fr !important;`

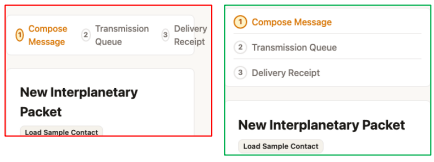
390 × 844 px · default	v3 · ep.01 · New rule created	trial-c51441fc2a0d
M06 rule-000005	**Constrain an invoice workspace to the viewport**	

Remove page-level horizontal overflow at every target viewport.
Example context
Invoice operations and decision workspace
	Before / After
	
	Scenario and filter controls shrink from an oversized panel to the phone viewport.
		Observed (before → after)
		Overflow (px) **299 → 0**
		Clipped elements **5 → 1**
		Residual: 8 overlaps, 1 clip. Regenerated synthetic data.
		CSS excerpt
		`.workspace-main`
		`min-width: 0;`
		`.workspace-sidebar-left`
		`min-width: 0;`

390 × 844 px · default	v4 · ep.70 · Retrieved rule	trial-66cfabca7c82
M07 rule-000043	**Expose the full artifact descriptions**	

Remove fixed-height clipping from meaningful copy and controls, then verify long labels and alternate states.
Example context
Museum of Lost Technology / results list
	Before / After
	
	The same two artifact cards show complete descriptions after the two-line limit is removed.
		Observed (before → after)
		Clipped elements **3 → 0**
		Removed CSS declarations
		`.card-desc`
		`- -webkit-line-clamp: 2;`
		`- overflow: hidden;`

390 × 844 px · default	v4 · ep.26 · New rule created	trial-480020257206
M08 rule-000048	**Keep every delivery step visible on a phone**	

Let content-driven regions grow and wrap while using scrolling only for explicitly wide, labeled data regions.
Example context
POST interplanetary message composer
	Before / After
	
	The clipped Delivery Receipt step becomes a full-width item in a vertical workflow.
		Observed (before → after)
		Overflow (px) **8 → 0**
		Clipped elements **1 → 0**
		CSS excerpt
		`.stepper-list`
		`/* below 650 px */`
		`flex-direction: column;`
		`align-items: stretch;`

390 × 844 px · default	v4 · ep.61 · New rule created	trial-9aef8864eee8

Figure A.3: **Memory lesson: layout boundaries and readable content.** Rule M05 (newly created), learned from a carbon-offset verification report: implement the phone mode with observable responsive behavior. Sidebars that squeeze the report are collapsed into a single column, restoring the title and metrics on a 390×844 viewport. Overflow drops from 251 px to 0, overlaps from 9 to 1, and clipped elements from 3 to 0; one overlap remains after the repair.

State feedback and semantic controls

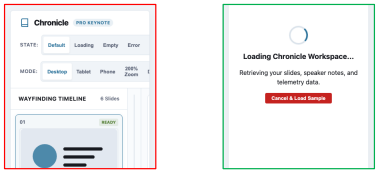
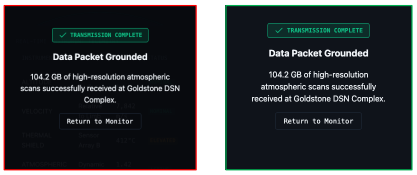
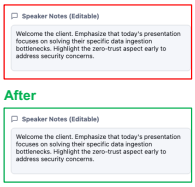
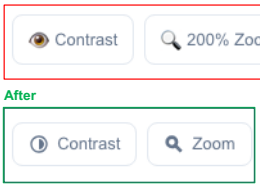
Memory element / rule	Visual evidence	Effect + source
M13 rule-000007	**Explicit loading status with recovery**	
Address missing loading state with this reusable principle: Model the condition as an explicit state with visible status, preserved input, recovery action, and an observable testing hook. **Example context** Speaker-notes workspace	Before / After  Loading-state captures: the workspace remains before; the repair adds visible progress and a recovery action.	**Observed (before → after)** Overlap count **7 → 0** Clipped elements **7 → 4** *Four clipping detections remain.* **CSS excerpt** `.state-loading .loading-overlay { display: flex !important; }`
390 × 844 px · loading	v3 · ep.10 · New rule created	trial-c0adaadd4aea
M14 rule-000041	**Give the mission confirmation its own readable region**	
Reflow narrow layouts so text, media, and actions remain fully visible without viewport-edge clipping. **Example context** ORBIT mission / transmission confirmation	Before / After  Success confirmation reflows into its own panel instead of overlaying telemetry.	**Observed (before → after)** Overflow (px) **30 → 0** Clipped elements **2 → 0** **CSS change, condensed** `.state-overlay - position: absolute; + min-height: 400px; + width: 100%;`
390 × 844 px · success	v4 · ep.24 · New rule created	trial-13199d2139b6
M15 rule-000021	**An explicit label for editable speaker notes**	
Address missing accessible label with this reusable principle: Express the correction through native semantics first, then add names, focus treatment, and non-color cues where needed. **Example context** Speaker-notes editor	Before / After  The notes editor remains visible; source evidence establishes the new label-to-field binding.	**Observed (before → after)** Notes label† **0 → 1** Notes fields† **1 → 1** *†Source counts; no assistive-tech replay.* **HTML excerpt, condensed** `<label for= "speakerNotesText"> Speaker Notes (Editable) </label>`
1440 × 1000 px · default	v3 · ep.24 · Stored evidence	trial-11781ced3bc9
M16 rule-000051	**A consistent icon system for interface controls**	
Replace emoji controls with a consistent inline SVG icon set, explicit dimensions, accessible names, and visible text where ambiguity remains. **Example context** Interactive paper-folding lesson	Before / After  Contrast and Zoom controls: emoji glyphs are replaced by explicit SVG icons.	**Observed (before → after)** Source emoji† **5 → 0** SVG elements† **2 → 10** *†Whole-source counts; includes hidden states.* **HTML excerpt** `<svg width="16" height="16" aria-hidden="true"> ... </svg>`
1440 × 1000 px · default	v4 · ep.83 · New rule created	trial-76919c3dada3

Figure A.4: **Memory lesson: state feedback and semantic controls.** Rule M13, learned from a speaker-notes workspace: model a missing loading condition as an explicit state with visible status, preserved input, a recovery action, and an observable testing hook. In the loading-state capture (390×844), the repair adds visible progress and a recovery action; overlaps drop from 7 to 0 and clipped elements from 7 to 4, with four clipping detections remaining.

Hierarchy and explicit task context

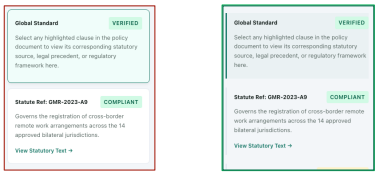
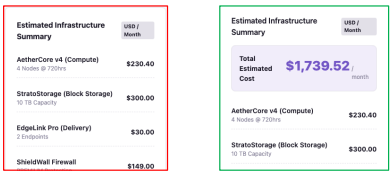
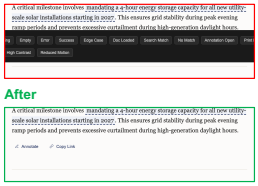
Memory element / rule	Visual evidence	Effect + source
M09 rule-000011	**A focal summary with a quieter evidence rail**	
Break the repeated-card silhouette with an asymmetric focal module, a quiet evidence rail, and borderless supporting content. **Example context** Policy and compliance viewer	Before / After  Matched citation-rail detail: separate boxed cards become continuous supporting evidence.	**Observed (before → after)** Visible cards 23 → 7 Overlap count 2 → 3 *Overlap detections increase.* **CSS excerpt** `.citation-card background: transparent; border: none; border-radius: 0;`
1440 × 1000 px · default	v3 · ep.13 · New rule created	trial-5458cbc4ab0c
M10 rule-000033	**Surface the primary cost finding**	
Use a deliberate typography ladder and tighter grouping so title, primary finding, evidence, and actions read in that order. **Example context** Infrastructure lead-capture form	Before / After  Summary-panel detail at different positions: total cost moves above the line items and gains visual emphasis.	**Observed (before → after)** Overflow (px) 129 → 0 Clipped elements 2 → 0 H1 elements† 0 → 1 *†Whole-source heading count.* **CSS excerpt** `.summary-primary-finding .total-price { font-size: 28px; font-weight: 800; }`
390 × 844 px · default	v3 · ep.28 · New rule created	trial-39c7f958d34e
M11 rule-000040	**Reflow auxiliary panels around document actions**	
Allow meaningful text and controls to wrap, grow, or reflow instead of being hidden by fixed dimensions or clipped overflow. **Example context** Grounded report viewer	Before / After  Matched detail: the fixed panel no longer covers Annotate / Copy Link.	**Observed (before → after)** Overlap count 19 → 0 Drawer clipping* 7 → 0 **Closed-drawer detections.* **CSS change, condensed** `.evaluator-panel - position: fixed; + margin-top: auto; /* drawer hidden */`
1440 × 1000 px · default	v4 · ep.81 · Provisional experiment	trial-5f4003804b53
M12 rule-000031	**Expose the default state in the header**	
Add a meaningful default state with visible status, recovery behavior, and an observable state marker. **Example context** Kintsugi product explorer	Before / After  Matched header: a state badge is added beside the Kintsugi identity.	**Observed (before → after)** State badge† 0 → 1 Reset button† 0 → 1 *†Source elements; reset not replay-tested.* **HTML excerpt, condensed** `<div id="state-indicator" data-state="default"> System: Default State </div>`
1440 × 1000 px · default	v4 · ep.09 · Stored evidence	trial-6fe70c03f9b1

Figure A.5: **Memory lesson: hierarchy and explicit task context.** Rule M09 (newly created), learned from a policy and compliance viewer: break the repeated-card layout with an asymmetric focal summary, a quiet evidence rail, and borderless supporting content. On a 1440×1000 desktop view, separate boxed cards become continuous supporting evidence, reducing visible cards from 23 to 7; overlap detections increase from 2 to 3, showing that a hierarchy repair need not improve every geometric check.



Figure A.6: **Additional memory-guided qualitative results across eight tasks.** From top to bottom: data annotation, analyst assistance, lesson delivery, design workspace, poster/banner creation, subscription management, system monitoring, and discussion forum. Columns show the input prompt, output without memory, retrieved rules, output with memory, and the dimensions with the largest score gains. Scores and repair-status labels are those recorded for each run. Screenshots show changes in interface organization; functional correctness and repair acceptance rest on execution evidence.

Table A.1: Task-specific verification requirements for the eight additional qualitative comparisons. Visible improvements must be accompanied by evidence that the corresponding interactions and data associations remain correct.

Case	Visible question	Required behavioral evidence
Data annotation	Are records, labels, and annotation controls clearly organized?	Selection and annotation edits remain attached to the intended record and preserve its associated data.
Analyst assistance	Are task context, actions, and tool state understandable?	Assistant actions respect the current tool state and required approvals while preserving the relevant task context.
Lesson delivery	Are lesson content, navigation, and progress easy to follow?	Navigation and progress updates remain associated with the correct lesson content.
Design workspace	Are the canvas, selected object, and property controls accessible?	Canvas edits preserve the selected object's identity and update the intended properties.
Poster/banner creation	Are composition controls and editable elements clearly presented?	Edits affect the intended element and preserve its content and properties through layout changes.
Subscription management	Are subscription details and management controls clearly grouped?	Subscription changes preserve the correct account and permission associations.
System monitoring	Are metric summaries, filters, and update states legible?	Filtering preserves metric identity, and delayed updates are handled without misrepresenting the displayed state.
Discussion forum	Are threads, messages, and participation controls easy to locate?	Forum actions preserve the intended thread, author, and message context.

B SAGE-CREATE: DESIGN CHOICES AND COMPLETE PROTOCOL

Section 1 started from a simple observation: when a generator's repairs are thrown away after each task, the same failures come back on the next one, and the user pays for each correction again. The question was how a verified repair can become knowledge that stops the generator from repeating the mistake. Answering it imposes four requirements, and each part of SAGE-CREATE exists to meet one of them.

1. *Know what must hold.* A change can only be called a repair, rather than a regression, if the task's obligations are explicit. *Reflect* compiles them into an experience contract (Appendix B.2).
2. *Verify by execution, including what must not break.* A lesson learned from a repair that silently broke something would spread the damage to every later task. *Refine* accepts a repair only when paired replay shows the target fixed and the protected behavior intact (Appendices B.3 and B.4).
3. *Store enough to reuse the repair correctly.* GUI failures are visual and behavioral, so *Remember* keeps the rule together with screenshots, the patch, and interaction traces (Appendix B.5).
4. *Decide which lessons deserve influence.* One success on one task is weak evidence, and a context window cannot hold every lesson. The *curriculum* lets lessons earn influence from support on distinct tasks and lose it when they stop helping (Appendix B.6).

For each component we describe what it does, why it is designed that way, and which experiment tests the choice.

B.1 PROBLEM SETTING: ADAPTING THROUGH MEMORY ALONE

Tasks arrive in sequence. Task l has input $u_l = (p_l, D_l, I_l, \mathcal{R}_l)$: a prompt, optional source data, optional visual references, and any structured requirements. Following Eq. equation 1, the generator returns the final GUI G_l and updated stores of verified lessons M_{l+1} and warnings from failed repairs

W_{l+1}, within a budget $\mathcal{B}$ and with model weights θ fixed. If no candidate passes the mandatory checks, it reports the unresolved requirements rather than returning an unverified interface.

Keeping θ fixed is a deliberate choice. Everything the system learns is an inspectable record: a rule, a diff, and the screenshots and traces that justified it. A lesson can be audited, retracted, or handed to a different model, which a weight update does not allow. The cross-model experiment tests the last property directly: a checkpoint learned with Gemini-3.5-Flash improves GPT-5 by +1.65 and Claude-4.8-Opus by +1.47 points, in all eleven families for both (Table 5). "Self-evolution" here therefore means that the content, scope, and priority of memory change; it does not mean that every new task improves.

B.2 Reflect: Why an Explicit Experience Contract

Most GUI defects are defined relative to the task. A table that overflows a phone screen is a defect only if the task requires a phone view; hiding a column fixes the overflow but breaks the task if that column holds the restore button. Without an explicit statement of what must hold, neither the verifier nor the memory can tell a repair from a regression. Reflect therefore compiles the request into the contract $\mathcal{C}_l$ of Eq. equation 2: the taxonomy category c_l, components K_l, states S_l, views V_l, and interaction tasks T_l, plus annotations for source bindings, style, accessibility, action permissions, and continuity. Each interaction task names an initial state, the actions, the expected outcome, and the properties that must persist. In a version-history interface, the selected record, the active filter, and the restore action must stay linked after the viewport changes. Supplied requirements are copied unchanged and assumptions are stored separately, so a later repair cannot quietly weaken the task.

As in the main text, the same contract plays three roles. It is the checklist that Refine executes, the query that retrieves lessons, and the context stored with every lesson learned on the task. Using one object for all three keeps them consistent: a lesson is retrieved for a task that shares the obligations under which it was verified. The records in Table B.1 link each obligation to the code that implements it and the check that tests it.

Record	Purpose
Evidence graph	Links supplied source records to visible claims, values, and components. Synthetic content is marked when no source is supplied.
Style contract	Records typography, color, spacing, hierarchy, and responsive constraints, including reference-specific requirements when available.
Interface program	Specifies component structure, layout, data bindings, and event handling independently of a particular front-end framework.
State machine	Records states, transitions, preconditions, side effects, confirmations, and required continuity.

Table B.1: **Records connecting the contract to generation and verification.** Shared identifiers connect requirements, implementation elements, and execution evidence.

Retrieving relevant lessons. Retrieval compares contracts, not prompts. Two prompts can describe the same need in different words ("the table breaks on mobile" and "records should be readable on a phone"), while their contracts share the same fixed vocabulary: a category from the taxonomy, component types, the six named views, the named states, and typed interaction tasks. For this comparison to work, every lesson must carry the contract under which it was verified. When Remember stores a repair, it therefore saves the full metadata of the task and of the repair episode with it (Table B.2).

Table B.2: **Metadata stored with every lesson and how retrieval uses it.** Contract fields come from Reflect on the task where the repair was accepted; episode fields come from Refine and Remember.

Field	Content	Use in retrieval
Category c_l	Family, subfamily, and leaf of the source task	Ranking; lessons from related patterns score higher
Task summary	Normalized user goal from the prompt	Ranking
Components K_l	Component types, e.g. table, form, chart, editor canvas	Scope filter and ranking
States S_l	Required states, e.g. loading, empty, error, success	Scope filter and ranking
Views V_l	Required views, and the view in which the failure appeared	Scope filter and ranking
Interaction tasks T_l	Action sequences, expected outcomes, properties that must persist	Ranking; tells the verifier what to re-check
Failure	Label (layout, interaction, state/recovery, data, accessibility, runtime), severity, affected component	Repair-time query and filter
Rule and patch	Repair rule r and the change π	Ranking (rule text); payload
Protected manifest	The obligations replay confirmed as preserved	Payload; re-verification on reuse
Support and status	Distinct supporting tasks, trial outcomes, lifecycle status, age	Eligibility filter and ranking

Retrieval then proceeds in four steps.

1. *Query.* Before generation, the query is the new task's contract: category, task summary, components, states, views, and interaction tasks. At repair time, the query also includes the diagnosed failure label, the affected component, and the view in which the failure occurred, so the lessons retrieved match the failure as well as the task.
2. *Filter.* A lesson is removed before ranking if its scope does not apply to the new contract, for example a phone-overflow lesson for a task with no phone view, or a table lesson for a task with no table; if an applicable warning contradicts it; if its evidence is missing; or if it is cooled or restricted. Candidate lessons are considered only when fewer than three active lessons apply, and at most two are admitted, at influence 0.25 instead of 1.0.
3. *Rank.* Query and lesson texts are compared with cosine similarity over L2-normalized unigram and bigram TF–IDF vectors fitted on the development lessons (no pretrained encoder), and lessons below 0.72 are dropped. The rest are ordered by a weighted score: 0.7 on similarity, 0.2 on the smoothed success rate over distinct tasks, and 0.1 on recency. Because the category is only one field among many, a lesson learned on one family can still be retrieved for another when components, views, and failure match. The audio-library overflow lesson of Figure 3, for instance, is relevant to the version-history task because both require a phone view and hold long-labelled rows.
4. *Assemble.* At most five lessons are passed on within 4,096 token equivalents, up to 800 per lesson. Each packet contains the rule, its scope, the relevant patch context, the target and protected observations, and image and trace excerpts, truncated in that order. If nothing passes the filters, retrieval abstains and generation proceeds without memory.

Storing the metadata with each lesson matters beyond matching. First, it bounds the lesson's validity: a repair verified on phone and desktop views says nothing about high-contrast mode, and the stored views make that explicit. Second, it lets the curriculum count support per context, so a rule is promoted for the kinds of tasks on which it actually succeeded, and a warning blocks it only where the same context recurs. Third, the stored interaction tasks and protected manifest tell the verifier which behavior to re-check when the lesson is reused on a new task. Matching on normalized

text keeps retrieval deterministic, cheap, and auditable; the multimodal evidence is carried in the payload, not used for matching.

Retrieving before the first candidate is written, not only after a failure, is also where memory saves cost. Iterative refinement pays for every defect with another generate–inspect–revise round, whereas a lesson applied up front can prevent the defect. The full pipeline uses 95.5k tokens and 6–8 minutes per GUI, against 114k tokens and about 22 minutes for five refinement rounds (Table 6). All N candidates receive the same contract and lessons; they differ in plan, layout, or component choice, never in their obligations.

B.3 Refine: Why Execution, Not Inspection

A screenshot can show that a button is visible but not that it works, and a model critic reading the code can believe a handler exists without it ever firing. Since many of the failures that matter are behavioral (a booking that does not submit, a selection lost on resize), Refine runs each candidate in a browser under the views, states, and interaction paths the contract requires and turns what it observes into failure evidence.

Execution setup. Candidates run in Chromium through Playwright, each in a fresh browser context. The contract selects its views from six rendering conditions: desktop (1440×1000), tablet (834×1112), phone (390×844), and 1280×900 with 200% text scaling, dark mode, or high contrast. A probe performs at most 15 actions with a 30-second timeout per action, and each candidate has a 120-second browser budget. Each probe stores its start state, actions, expected result, and required captures, so it can be replayed exactly after a patch. Every observation (frame, element geometry, DOM and accessibility tree, focus and action trace, console log, application state) is tagged with the candidate, requirement, view, and capture time.

Rendering and runtime checks. Table B.3 lists the checks run on every rendered view and the evidence each failure produces. Geometric and behavioral checks are deterministic assertions over the DOM and application state. The online critic (Gemini-3.5-Flash, temperature 0) reviews screenshots for hierarchy, grouping, and readability, but its findings only propose repair targets; a repair counts only if one of the measured checks confirms the fix.

Table B.3: **Checks run in Refine and the failure evidence they record.** All checks run in every view the contract requires; tolerances are fixed before execution.

Check	What is measured	Evidence kept on failure
Build and render	Page loads without uncaught exceptions, console errors, or requests to external resources; the root renders non-empty content.	Console and runtime log; screenshot.
Overflow	Document scroll width against viewport width; element boxes extending past the viewport or past an ancestor that hides overflow (1 CSS px tolerance).	Element geometry; overflow in px; screenshot crop.
Clipping and truncation	Required text or controls cut off by a clipping ancestor, or text whose content height exceeds its box.	Element and ancestor boxes; crop.
Overlap	Intersecting boxes of interactive elements; the element at a control's click point is not the control itself.	Both boxes; occluding element.
Required components	Each component in K_l exists, has non-zero size, is not hidden, and is on screen or reachable by scrolling.	DOM node or its absence; screenshot.
State reachability	Each state in S_l (loading, empty, error, success, edge) is reached through its prescribed action path and shows its expected content and controls.	Action trace; DOM snapshot; screenshot.
Interaction outcome	Each task in T_l is executed; the expected DOM and application-state change occurs, and properties that must persist (selection, filter, entered values) survive view changes.	Action trace; state before and after.
Data consistency	Displayed values match application state and fixtures, e.g. totals equal the sum of line items (IDs and currency exact in integer minor units; other values within 10^{-6}).	Expected and displayed values.
Accessibility	Controls have accessible names; keyboard traversal reaches every required control; focus is visible; text and controls meet WCAG 2.2 contrast minima (4.5:1 for text, 3:1 for large text and UI components).	Accessibility tree; focus trace; contrast ratios.

From checks to failures. Each check returns *pass*, *issue found*, or *not observed*. An issue becomes a failure record f with a label (layout, interaction, state/recovery, data, accessibility, or runtime), a severity, the affected component and source region, the view in which it appeared, and the IDs of its evidence. A failure is blocking when it is a startup or runtime failure, an unreachable primary action or recovery path, or a prohibited destructive action. A repair is attempted when there is a high or critical finding or when the verifier's score falls below 0.92. Coverage is whatever the probes exercise; we do not claim to enumerate every reachable state.

The controls support this choice. Without memory, the generator already reaches 72.04, above Self-Refine (69.2) and Best-of-3 sampling without a verifier (70.5), which revise or sample without executing the interface (Table 3). With four candidates and the same one-patch budget, the generator also beats Best-of-4 + repair by 0.69 points, so the gain comes from grounding decisions in execution rather than from more samples.

B.4 Why Repairs Must Preserve, Not Just Fix

A repair judged only on its target is easy to game, often unintentionally. Clipping overflow hides the overflow; removing a failing widget removes the failure. Accepted locally, such a patch costs one interface. Stored in memory and retrieved on dozens of later tasks, it becomes a systematic regression. The acceptance rule is therefore also the gate into memory, and it checks preservation as strictly as resolution.

Before editing, the generator picks a failure f and fixes a probe manifest $R = R_{\text{target}} \cup R_{\text{protected}}$. Target probes reproduce the failure; protected probes cover the content, actions, accessibility, runtime validity, and user state that must survive. The generator records baseline evidence B, proposes one focused patch π that tests a single failure hypothesis, keeps the inverse patch, and replays $G' = \pi(G)$ on the same manifest to obtain A. Both runs start from the same data, user state, viewport, and mode, with resettable fixtures, because reverting source code alone does not reset application state. The repair is accepted only when

$$\begin{aligned}\text{Accept}(f, R, B, A) = \; & \text{Complete}(R, B, A) \wedge \text{Resolved}(f, A) \wedge \text{Improved}(f, B, A) \\ & \wedge \text{Valid}(A) \wedge \text{Preserved}(R_{\text{protected}}, B, A) \wedge \neg\, \text{NewSevere}(B, A).\end{aligned} \tag{B.1}$$

Complete requires every mandatory probe and capture in both runs; *Resolved* and *Improved* require the target to pass and its measurement to improve; *Valid* requires the interface to build and render; *Preserved* requires every protected obligation to hold; and the last term rules out a new high-severity failure. Tolerances are fixed before replay, and measurements keep their own units (pixels of overflow, pass/fail for a transition). Before each of the two runs, the fixture, route, viewport, clock, cookies, and browser storage are restored in a fresh browser context, and the target overflow must fall by at least 1 px.

Two design details follow from the same concern. First, missing evidence is never read as success: it yields *inconclusive*, and like *reject* it restores the previous candidate and creates no lesson. Second, rejected repairs are kept as warnings in W. A warning records that a particular fix broke something in a particular context, so the same fix is not retried there and cannot be promoted by later retrieval.

B.5 Remember: Why Multimodal Memory

Memory in prior agents is usually text, such as the verbal reflections of Reflexion (Shinn et al., 2023); executable skill libraries such as Voyager (Wang et al., 2023) store code instead. For GUIs, either alone loses something essential. A rule such as *"keep controls within the viewport"* does not say what the failure looked like, how it was fixed in code, or whether the fix kept the interface working. A diff says how but not why, and it does not transfer to a page with different markup. What a later task needs is the whole repair episode.

Each lesson therefore stores the record m of Eq. equation 3: the applicability context x (the contract), the failure f, the reusable rule r, the before and after evidence E^-, E^+, the patch π, and the history h. In the implementation it also keeps the measured effect δ on target and protected checks and a lifecycle status z. Distillation writes the lesson as *trigger* $\rightarrow$ *failure* $\rightarrow$ *rule* $\rightarrow$ *measured effect* and drops page-specific selectors while keeping links to the original evidence. Table B.4 lists what each modality contributes.

Table B.4: **A multimodal lesson connects what changed to why the change is reusable.** Evidence is aligned by candidate, requirement, element, execution context, and capture time. The modalities play complementary roles in supporting a repair rule.

Memory content	Evidence retained and role in reuse
Semantic rule	Context, trigger, failure, and generalized repair explain when and how the lesson applies.
Visual evidence	Before/after frames and affected regions show overflow, hierarchy, clipping, and the observed visual correction.
Program evidence	Changed source, inverse patch, element geometry, and document/accessibility structure connect a visible defect to its implementation.
Behavioral evidence	Actions, focus traces, reached states, and application state show whether interactions and user context survived the repair.
Verification history	Probe manifest, measured effects, distinct-task support, and warnings delimit the evidence for later retrieval.

The modalities are complementary, not redundant, and each serves a different reader. For the *generator*, a before/after crop and a diff ground the rule. They show what "overflow" meant in that case and one concrete way to fix it, which is easier to apply faithfully than a sentence. For the *verifier*, the stored trace and manifest say which behavior the original repair had to preserve, so the same

protected checks can be run when the lesson is reused. For the *curriculum*, the patch makes credit assignment possible: a lesson is credited only when a later patch actually follows its rule, which can be checked against the stored change. The audio-library lesson in Figure 3 is typical. The screenshot shows labels spilling past the viewport, the patch shows the wrapping and grouping, and the trace shows that every secondary control is still reachable.

The ablation isolates this effect. With the same generator and budget, adding text rules improves the score from 72.04 to 72.48, and attaching the multimodal evidence raises it further to 73.01 (Table 3), a larger step than the text alone. The evidence must also be *task-specific*. A fixed twelve-item checklist of good GUI practice, given the same prompt slots and token allowance, recovers only 0.57 of the 2.08-point memory gain (Table 4). Finally, admission uses exactly Eq. equation B.1, so a good-looking after-image or a favorable critic cannot create a lesson on its own.

Selective retrieval. Retrieval is selective and may abstain, in the spirit of selective prediction (Geifman & El-Yaniv, 2019): an irrelevant lesson distracts the generator and uses context that a relevant one needs. The metadata, filters, and ranking are described in Appendix B.2. A text-only generator receives textual descriptions of the evidence, and the verifier keeps the original evidence either way. Every reused repair is verified again on the current task.

B.6 Curriculum: Why Lessons Must Earn Influence

A bank that stores every accepted repair and retrieves by similarity alone runs into four problems. It fills with near-duplicates of the same fix, so retrieval returns five versions of one lesson. It treats a repair that worked once as a rule, although a single success is weak evidence that the rule generalizes. It keeps stale or harmful lessons active indefinitely. And because the context budget is fixed, it cannot give priority to the lessons that have actually helped. The curriculum addresses each of these by tying a lesson's influence to verified evidence across distinct tasks (Table B.5).

Table B.5: **Memory learning curriculum.** Each stage changes how much influence a lesson has, based on verified evidence; generator weights stay fixed. Thresholds are given in the text.

Stage	What happens	Why
Explore (creation)	An accepted repair that follows an existing rule adds support to that lesson. A repair with a new rule enters as a low-influence candidate and is trialed only when few active lessons apply.	The bank grows with new failure types rather than repeated fixes, and new rules can gather evidence without steering generation.
Consolidate (promotion)	A candidate becomes active once accepted repairs on several distinct tasks give it a high success rate, unless an applicable warning contradicts it.	Support across tasks, not repeated success on one, is what suggests that a rule generalizes.
Retain (pruning)	Unused lessons are cooled, repeatedly failing lessons are restricted, and at capacity the lowest-ranked inactive lesson is evicted. Cooled lessons return after a verified trial.	The active pool stays small and current within a fixed context budget; retired lessons keep their evidence for audit.

Two rules keep this evidence honest. Support is counted per task, using a stable task identifier d, so repeated probes on one task cannot inflate a lesson's record. And a retrieved lesson is credited or charged only when the applied patch actually follows its rule within its scope. A successful patch that does something else becomes a separate candidate, and inconclusive trials count neither way. A joint patch that draws on several lessons cannot attribute its effect to any one of them, so Algorithm B.1 updates lessons one repair at a time. The stages saves experience specific metadata with lessons accumulate evidence, so a new kind of task can still be exploring while another is mature.

Thresholds. A candidate is promoted to active after 3 accepted trials on distinct tasks with smoothed success rate $p = (s + 1)/(s + f + 2) \geq 0.75$ and no applicable warning; active and candidate lessons have retrieval influence 1.0 and 0.25. A lesson unused for 20 development tasks is

cooled, and one with a new severe failure, or 2 rejects among its last 5 conclusive trials, is restricted; a restricted rule can return only as a revised rule with a new ID. Every 5 development tasks, one applicable cooled lesson is re-trialed under the normal acceptance rule and reactivated if accepted.

Algorithm B.1 One repair attempt and its memory update

Require: Task ID d, contract $\mathcal{C}$, candidate G, failure f, memory M, warnings W, policy $\mathcal{P}$
$R \leftarrow \textsc{SelectProbes}(\mathcal{C}, f)$
$B \leftarrow \textsc{Execute}(G, R)$
$\ell \leftarrow \textsc{RetrieveOrAbstain}(M, W, \mathcal{C}, f, \mathcal{P})$
$\pi \leftarrow \textsc{ProposePatch}(G, \mathcal{C}, f, \ell, W)$; store inverse π^{-1}
$G' \leftarrow \pi(G)$; $A \leftarrow \textsc{Replay}(G', R)$
$o \leftarrow \textsc{CheckAcceptance}(f, R, B, A, \mathcal{C})$
$\delta \leftarrow \textsc{RecordEffect}(R, B, A)$
if $o = \text{accept}$ **then**
    $u \leftarrow \textsc{DistillLesson}(\mathcal{C}, f, \pi, \delta, B, A)$
    $M \leftarrow \textsc{RecordSupport}(M, u, \ell, d)$; $G \leftarrow G'$
else
    $G \leftarrow \textsc{Rollback}(G', \pi^{-1})$
    $M \leftarrow \textsc{RecordTrial}(M, \ell, d, \mathcal{C}, f, \pi, o, \delta)$
    if $o = \text{reject}$ **then**
        $W \leftarrow \textsc{AddWarning}(W, \mathcal{C}, f, \pi, B, A)$
    end if
end if
$M \leftarrow \textsc{UpdateLifecycle}(M, W, \mathcal{P})$
return G, M, W

Three results support the curriculum. It gives the largest single step in the ablation, +1.11 on top of multimodal memory (Table 3 in main paper). The gain is not simply more memory: with the same 256-lesson pool and the same number of retrieved lessons per task, the curriculum beats similarity-only retrieval by 0.94 points. And its advantage grows as evidence accumulates, from 0.18 points at the first development checkpoint to 0.99 at the last. The bank grows from 33 to 51 lessons while the eligible pool stays between four and ten. New rules per episode fall from 0.50 to about 0.17, consistent with recurring failures strengthening existing lessons instead of adding new ones.

C SAGE-BENCH: TAXONOMY, CONSTRUCTION, AND TASK CONTRACTS

SAGE-CREATE is meant to improve an interface without breaking the user's task, so the benchmark has to test the whole task, not just a well-rendered first screen or one working button. A report viewer can look right and support search, yet lose the selected citation after a resize or have no way back from an error. Existing benchmarks cover important parts of this: screenshot reconstruction (Si et al., 2025), interactive reconstruction (Xiao et al., 2024), instruction-conditioned website functionality (Lu et al., 2026b), and dynamic artifacts and development workflows (Zhang et al., 2025a; He et al., 2026). SAGE-BENCH adds a single task specification that ties the user's goal to required content, reachable states, usable views, working actions, and continuity across transitions. Different layouts can satisfy the same specification, so no single reference screenshot defines the answer.

The active release has **300 prompt-only tasks**, one per taxonomy leaf, each asking for a complete executable web interface. The general schema also allows source documents and visual references; both are empty in this release.

C.1 EXPERIENCE TAXONOMY AND COVERAGE

Tasks are organized first by *user intent*, then by the workflow that serves that intent, and finally by the interface pattern that realizes it. The design draws on task abstraction, which separates purpose from inputs and interaction (Brehmer & Munzner, 2013), and on guidelines for human control, correction, and behavior over time (Amershi et al., 2019). Accessibility requirements follow WCAG 2.2 (W3C Accessibility Guidelines Working Group, 2023) and apply to every family. These sources guided the design; the hierarchy itself is ours.

The hierarchy has **11 experience families, 48 subfamilies, and 300 path-qualified leaves** (Table C.1). A leaf is a concrete experience such as a report viewer, a checkout flow, a code-review workspace, a scientific figure, or an agent-governance console. Qualifying leaves by their path separates patterns that share a name in different contexts. Each leaf contributes one task, so family sizes reflect how many distinct patterns a family contains, not balanced sampling.

Table C.1: Complete SAGE-BENCH taxonomy at the family and subfamily levels. Parentheses give leaf/task counts. Each of the 300 leaves contributes one prompt-only task; all 48 subfamilies are represented.

Experience family	Subfamilies (tasks)	Total
Information	Document Consumption (7); Search & Discovery (7); Monitoring (7); Reporting (7); Knowledge Exploration (7)	35
Operational	Transaction Processing (7); Workflow Execution (7); Case Management (7); Administration (7); Safety, Security & Compliance Operations (6)	34
Decision	Analysis (7); Comparison (7); Planning (7); Forecasting (6); Recommendation (6); Optimization (6)	39
Creation	Content Creation (7); Data Creation (7); Asset Creation (7); Configuration Creation (7)	28
Collaboration	Communication (7); Review (7); Coordination (7); Shared Workspace (7)	28
Learning & Assessment	Learning Content (6); Assessment (5); Practice & Simulation (5); Curriculum & Progress (5)	21
Presentation & Narrative	Slide Decks (5); Storytelling Layouts (5); Carousel and Card Narrative (4); Speaker and Audience Support (5)	19
Commerce & Growth	Merchandising (5); Conversion (6); Campaigns (6); Customer Lifecycle (5)	22
Data & Scientific Communication	Visualization (6); Scientific Figure (5); Data Exploration (5); Explainability (5)	21
Agent	Agent Assistant (7); Agent Workspace (7); Multi Agent Workspace (7); Agent Operations Center (8)	29
System & Device	Device UI (7); Settings & Control (6); Installation & Setup (5); Environment Specific UI (6)	24
11 families	**48 subfamilies**	**300**

Relationship to existing task forms. Table C.2 maps the task forms of representative benchmarks onto the contract fields. Visual reconstruction constrains content and appearance; interactive reconstruction adds states and transitions; instruction-conditioned generation specifies intended outcomes. Writing all of them in one contract shows how they relate while keeping their different goals. The taxonomy also brings collaboration, learning, scientific communication, agent supervision, and system control into the same hierarchy. The mapping is conceptual. We do not claim that the 300 tasks subsume these benchmarks item by item, and full-stack services and native device execution are outside the current release.

Table C.2: Task-form coverage and its scope. The last column is our mapping of published task forms into SAGE-BENCH, not a measured coverage rate. p, D, I, K, S, V, T denote prompt, source assets, visual references, components, states, views, and interaction tasks. Reference-conditioned mappings require the optional I field; the active release uses text only.

Benchmark	Evaluation focus	Corresponding SAGE specification
Design2Code (Si et al., 2025)	Screenshot-to-code fidelity	I, K, V: content and presentation requirements in Information, Presentation, and Commerce patterns.
Interaction2Code (Xiao et al., 2024)	Code from interactive prototypes	I, K, S, T: visible transitions and action outcomes in Operational and other interactive patterns.
WebGen-Bench (Lu et al., 2026b)	Websites from instructions; expected action results	p, K, S, T: content presentation, user interaction, and data management across Information, Operational, Creation, and Data patterns.
DesignBench (Xiao et al., 2025b)	Frontend generation, editing, and repair	K, S, V, T: requirements for the resulting interface; framework-specific development tasks are separate from our task taxonomy.
ArtifactsBench (Zhang et al., 2025a); UI-Bench (Jung et al., 2025)	Dynamic artifact quality; expert visual preference	p, K, S, V, T: visual and interaction requirements across Creation, Presentation, Learning, and other experience families.
Vision2Web (He et al., 2026)	Static, interactive, and full-stack website development	I, K, S, V, T: frontend appearance and workflow obligations; backend implementation is outside our active protocol.
WebArena; VisualWebArena (Zhou et al., 2024; Koh et al., 2024)	Agent task completion on existing websites	S, T: task outcomes in Commerce, Collaboration, and Information patterns; SAGE evaluates the generated GUI, not the navigating agent.
SAGE-BENCH	**Complete generated GUI experiences**	**11 families, 48 subfamilies, 300 tasks**; components, recovery states, views, actions, accessibility, and task continuity.

C.2 FROM TAXONOMY LEAVES TO EXECUTABLE CONTRACTS

For each leaf, a deterministic planner derives a scenario seed from the leaf's identifier and name and assigns a target user, an artifact, use cases, domain terms, and one edge condition. Prompt and context builders turn this plan into a natural-language request and structured requirements (Figure C.1). Every system receives the same text bundle: prompt, taxonomy path, planning context, contract, design and interaction context, and output requirements. Generators therefore see the obligations they will be judged against, but never the evaluator's output.

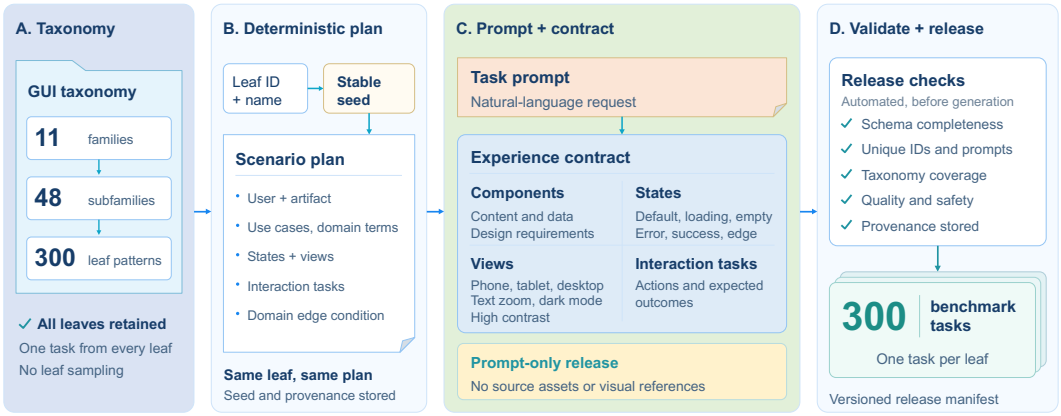


Figure C.1: Construction of SAGE-BENCH. Each taxonomy leaf receives a deterministic scenario plan, which is compiled into a prompt and structured component, state, view, and interaction annotations. Validation checks completeness, uniqueness, taxonomy coverage, and provenance before assembling the 300-task evaluation pack. Source and reference fields are empty in the active release.

A benchmark instance b_l bundles the prompt p_l, source documents or data D_l, visual references I_l, and the component, state, view, and interaction sets K_l, S_l, V_l, T_l. These are the same sets that SAGE-CREATE compiles into its contract (Eq. equation 2), with annotations for accessibility, data consistency, domain policy, and continuity. In this release $D_l = I_l = \varnothing$ for every task, so any data the interface shows is synthetic. It must be coherent and disclosed, and the release cannot test fidelity to an external document or design.

Every task requires default, loading, empty, error, success, and a domain-specific edge state, and six views: phone, tablet, desktop, 200% text zoom, dark mode, and high contrast. Task annotations prescribe actions and their expected outcomes, covering state reachability, responsive behavior, and data consistency, with extra workflow-specific interactions for patterns such as dashboards and reviews. Components count only when they are usable parts of the experience; keyword presence earns nothing. Table C.3 shows how the same kinds of annotation take on different meanings across families.

Table C.3: Representative contracts, summarized from benchmark cases. All share the state/view requirements described in the text; the task-specific components and failure conditions distinguish their user experiences.

Family / leaf	Components and user goal	Interaction and recovery obligations
Information / KPI Dashboard	Metric tiles, trends, severity filters, and an alert timeline support monitoring.	Filtering updates visible signals; values remain consistent across tiles and charts; handle a delayed metric stream or alert spike.
Operational / User Management	User table, permissions, settings, and audit history support administration.	Provide permission feedback and reachable recovery states when a restricted action is attempted.
Collaboration / Code Review	Difference view, comments, review decisions, and version history support coordinated review.	Distinguish approval from requested changes; update decision history and handle unresolved or conflicting feedback.
Agent / Agent Governance	Agent roster, plan, tool-call log, and approval checkpoint support human supervision.	Expose permission risks, preserve action feedback, and make escalation available when a tool request exceeds authority.

C.3 Generation Protocol and Release Integrity

Every output is a single HTML document with embedded CSS and JavaScript that runs directly in a browser. External scripts, fonts, images, services, and CDNs are not allowed, so artifacts can be archived and replayed offline. Direct model baselines receive the same instructions and case bundles. An orchestration system must disclose how it translates the prompt, which fields it drops or adds, and its internal generation stages. For each run we keep the supplied bundle, the final HTML, the model response or orchestration report, and generation metadata. Decoding settings, budgets, and retry policies are fixed per experiment and reported with it.

Release validation checks schema completeness, uniqueness of identifiers and prompts, taxonomy coverage, planning diversity, provenance, and content constraints. The released pack has 300 distinct task identifiers covering all 11 families and 48 subfamilies, with all seven instance fields present. This establishes structural coverage; whether the scenarios saturate each leaf semantically would need an independent expert mapping. Benchmark tasks are used only for evaluation: memory is built on separate development tasks, and every evaluation task starts from the same frozen checkpoint. Reports account for all 300 tasks, including missing or invalid generations, and give both task-weighted and family-level summaries. Because each leaf contributes one task, these summaries describe the declared taxonomy rather than the population of all possible GUI requests. Any change to prompts, contracts, seeds, execution settings, or evaluator versions defines a new benchmark configuration.

C.4 Comparison with Existing GUI-Generation Benchmarks

Table C.2 mapped task forms; Table C.4 compares nine benchmarks of generated web interfaces or visual artifacts in more detail, separating dataset design (panel a) from the evidence each evaluation protocol uses (panel b). Counts keep their original units, since a webpage, a generation request, an edit/repair instance, and a browser test are different objects. We use conference versions for Design2Code, Interaction2Code, WebGen-Bench, and WebCoderBench; UI-Bench v3; and the v1 preprints of DesignBench, ArtifactsBench, Vision2Web, and WebCompass. For Interaction2Code this means the published release (127 pages, 374 interactions) rather than the smaller 2024 preprint. WebArena and VisualWebArena evaluate agents operating existing sites, so they are left out of this generation comparison (Zhou et al., 2024; Koh et al., 2024).

What distinguishes SAGE-Bench is that every task contract jointly specifies components, reachable and recovery states, presentation conditions, actions, and continuity obligations, and that SAGE-Eval requires evidence for each. None of these aspects is new on its own. WebGen-Bench tests action outcomes; Vision2Web includes responsive and full-stack development; WebCoderBench measures mobile overflow and accessibility; and WebCompass tests preservation during editing and repair as well as generation. Our release is also smaller than several of these, uses constructed text-and-contract tasks without external assets or backend services, and its evaluator has been validated against expert ratings on 88 tasks (Appendix E) rather than at scale. The table positions the protocol; it does not claim an advantage in every capability or a measured coverage rate.

How coverage is classified. *Action outcomes* require executed actions with checked results: ArtifactsBench works from code and temporal captures, and WebCoderBench's functional judgments are based on code. *State/recovery* separates a prescribed set of states from selected transitions, prompt requirements (UI-Bench), or code checks. *Views/modes* separates a fixed matrix of devices and modes from device-only or task-specific checks; Vision2Web specifies three devices at level 1, and WebCoderBench audits mobile overflow. *Accessibility* means selected audits, not full standards conformance; DesignBench and WebCompass include relevant repair criteria. *Data/continuity* separates explicit preservation of values and identity through interaction and recovery from task-specific functional or preservation tests. WebCompass uses browser-agent testing for generation and judges edits and repairs from patches, logs, and before/after renders.

(a) Task design, evaluation scale, and human involvement

Benchmark / source	Evaluation set	Input and task	Provenance and coverage	Human involvement
Design2Code (Si et al., 2025)	484 pages	Screenshot to webpage code	Real webpages from C4	Manual curation; output ratings
Interaction2Code (Xiao et al., 2025a)	127 pages; 374 interactions	Prototype images and actions to interactive code	Real pages; 15 page types, 31 interaction categories	Interaction annotation; output ratings
DesignBench (Xiao et al., 2025b)	900 instances: 430 G / 359 E / 111 R	Images, code, text; generation, editing, repair	Real/translated examples; 4 code stacks, 11+ topics	Expert curation; edit/repair judge validation
WebGen-Bench (Lu et al., 2026b)	101 requests; 647 tests	Text to multi-file websites	Human/LLM-authored; 3 main, 13 minor categories	Test review; agent–human agreement
ArtifactsBench (Zhang et al., 2025a)	1,825 tasks	Text to executable visual artifacts	Curated/LLM-expanded; 9 topics	Checklist review; judge validation
UI-Bench (Jung et al., 2025)	30 prompts; 300 generated sites	Text to websites and web apps	Synthetic client briefs; 5 categories	4,000+ expert pairwise judgments
WebCoderBench (Liu et al., 2026)	1,572 requests	Text; optional images/URLs; web-app generation	Real user requests; 20 application categories	Expert filtering; preference-based metric weights
Vision2Web (He et al., 2026)	193 tasks; 1,255 tests	Images + text; static to full-stack	Real sites; 3 task levels, 16 subcategories	Expert workflows; verifier/judge validation
WebCompass (Lei et al., 2026)	1,526 tasks: 326 G / 600 E / 600 R	Text, images, video, code; generation, editing, repair	Mixed sources; 15 domains, 16 edit / 11 defect types	Expert review; judge validation
SAGE-BENCH (ours)	**300 tasks**	**Text + contract to self-contained GUIs**	**Constructed; 11 families, 48 subfamilies**	Structural checks; 20-expert validation (App. E)

(b) Evidence and scope of the associated evaluation protocol

Benchmark	Visual assessment	Action outcomes	State / recovery	Views / modes	Accessi-bility	Data / continuity
Design2Code	✓	NR	NR	NR	NR	NR
Interaction2Code	✓	✓	**P**	NR	NR	NR
DesignBench	✓	NR	NR	NR	**P**	NR
WebGen-Bench	✓	✓	**P**	NR	NR	**P**
ArtifactsBench	✓	**P**	**P**	NR	NR	NR
UI-Bench	✓	NR	**P**	NR	NR	NR
WebCoderBench	✓	**P**	**P**	**P**	✓	NR
Vision2Web	✓	✓	**P**	**P**	NR	**P**
WebCompass	✓	✓	**P**	**P**	**P**	**P**
SAGE-EVAL (Ours)	✓	✓	✓	✓	✓	✓

Reading the table. ✓ = explicit criterion supported by the specified evidence; **P** = partial, task-specific, or proxy coverage; NR = no dedicated evaluation reported, not proof of absence. WebCompass browser-agent testing applies to generation. Symbols describe protocol scope, not performance or complete validation. Coverage criteria are explained in the text.

Table C.4: **Positioning SAGE-BENCH among GUI-generation benchmarks.** Dataset design and scale (a), followed by evidence and coverage (b). Panel (b) is our classification of the cited protocols, not an empirical re-evaluation. Checkmarks describe specified evidence requirements, not measured success or comprehensive validation.

D SAGE-EVAL: COMPLETE METRICS AND EVIDENCE PROTOCOL

SAGE-BENCH says what the user should be able to see and do; SAGE-EVAL measures how well a frozen interface delivers it. Some failures only show up at the level of the whole experience. A checkout can look right and still lose the order after an error, and a working dashboard can hide the relationships its user needs. We therefore propose three **experience-quality metrics**: **perceptual alignment** (is the visible journey coherent and suited to the task?), **visual layout** (is the content legible and usable in every required view?), and **data integrity** (do values and relationships stay correct through interaction?). They sit alongside eight adopted dimensions for contract fidelity and execution reliability. Visual and interactive judging already appear in ArtifactsBench and executable website tests in WebGen-Bench (Zhang et al., 2025a; Lu et al., 2026b); our formulations tie each judgment to explicit contract requirements and observable evidence. Every metric returns a score in $[0, 100]$ together with the checks that passed, localized failures, and the evidence behind them. The metrics overlap when a single defect affects several aspects of the experience, and reporting them separately keeps that visible.

SAGE-EVAL compares the frozen interface G_l with the requirements of its benchmark case (Appendix C.2) without assuming a unique reference design. Evaluation starts only after generation and selection have finished, and its scores, judge responses, and diagnostic reports never reach generation, repair, or memory. The verifier inside SAGE-CREATE decides whether a repair is accepted; SAGE-EVAL scores the final output of any generator. Figure D.1 summarizes the evidence flow.

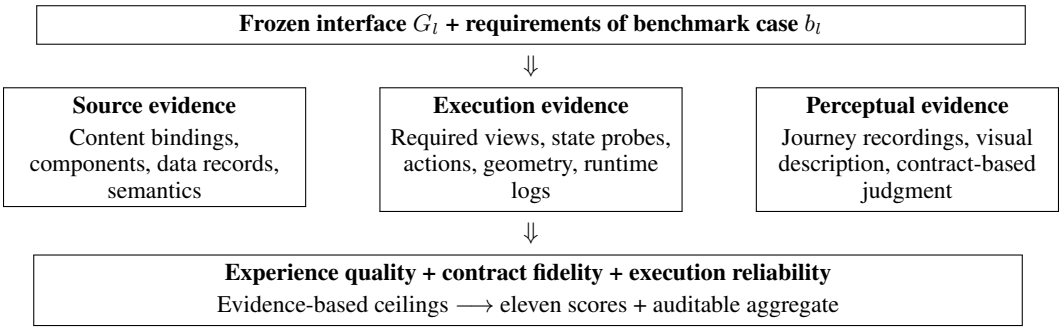


Figure D.1: SAGE-EVAL collects complementary evidence about a frozen GUI. Its proposed experience-quality metrics assess perceptual alignment, visual layout, and data integrity; adopted dimensions assess contract fidelity and execution reliability. Observed failures constrain scores, including visual judgments. External evaluation results do not feed the SAGE-CREATE learning loop.

D.1 PROPOSED EXPERIENCE-QUALITY METRICS

Perceptual alignment. An interface should communicate its task throughout use, including feedback, transitions, and changes of view. The evaluator records sessions that scroll, inspect overflow, and exercise task controls and state selectors. A video-capable model first describes the visible content, hierarchy, visual primitives, transitions, responses to interaction, and defects. A second call scores that description against the contract. The rubric covers fit to the artifact type, how well the task and its data are communicated, feedback, and coherence across the recorded states and conditions. Splitting description from judgment makes the observations behind a score inspectable. Polish cannot show that a control works, so the judged score is capped by an execution audit: observed contract violations, failed controls, missing states, or rendering defects each set a maximum (Section D.3), and the final score is the lower of the judgment and that cap. Each applied ceiling is logged with its evidence.

Visual layout. This metric asks whether the rendered interface keeps the required content and actions legible and usable in every view in V_l. It covers hierarchy, grouping, readability, control visibility, and the visual mapping of task information, along with geometric defects such as overlap, clipping, and overflow. A responsive CSS rule is supporting evidence only; what counts is its rendered effect. Each view in V_l gets a score from a predeclared, weighted set of rubric checks, and the metric is the average over views. Hierarchy and readability are judged from screenshots; clipping,

overlap, and overflow are computed from element geometry. A missing render gets no credit, and severe failures trigger further ceilings. Visual layout diagnoses presentation view by view, whereas perceptual alignment judges the journey as a whole.

Data integrity. This metric asks whether values and relationships remain correct as the interface is used, edited, and recovered. Its evidence links source or declared initial records to application state, displayed values, transformations, and action outcomes. A checkout total, for instance, must match the selected items, update after a quantity change, and still match after an error and retry; a graph must keep the relationships in its underlying records. These obligations are fixed before execution, not inferred from a plausible output. The metric is the weighted share of applicable data obligations, derived from the input and contract, for which the evidence shows the obligation holding across its prescribed actions and recovery states. An obligation whose required evidence is missing scores zero, and an observed failure receives its rubric value. When D_l is supplied, source records allow checks of factual fidelity. In the current prompt-only release the evaluator checks the prompt's facts and constraints and the internal consistency of declared synthetic records, transformations, and displayed results; it makes no claim about external factual grounding. An interface with no applicable data obligations is marked not applicable (Section D.3), which is different from a required check that cannot be verified and therefore scores zero.

D.2 Adopted Dimensions and Evidence Collection

Contract fidelity adopts five dimensions, each measured against the SAGE-Bench contract so that different designs can satisfy the same task. Prompt alignment checks that the requested goal, audience, and content are addressed; component coverage that required components are present and usable; state coverage that required states are reached with the expected content and controls; view coverage that every prescribed viewport, zoom, and mode keeps content and actions usable; and task support that actions produce their expected results with the required context intact. *Execution reliability* adopts three: build and render (the interface loads and runs without blocking errors, judged from build output, route checks, and runtime logs), accessibility (semantics, labels, keyboard navigation, focus, and contrast), and safety, licensing, and provenance (permissions, confirmations, and traceable content and assets). These dimensions are the context for reading the experience metrics: a coherent-looking interface still fails its contract if a required action does nothing.

Source inspection parses HTML, CSS, JavaScript, and any data bindings. Browser execution collects rendered views, component inventories, state probes, action outcomes, accessibility structure, and runtime logs, using Chromium through Playwright under six conditions: desktop (1440×1000), tablet (834×1112), phone (390×844), and 1280×900 renders with 200% text scaling, dark mode, and high contrast. Journey recordings for perceptual alignment cover desktop, phone, and dark mode. The contract's actions determine which states, data transformations, and continuity requirements must be observed.

Static hooks do not count as successful execution. A state label does not show that the state is reachable, and an event handler does not show that the action produces the expected result. Accessibility checks likewise test selected requirements, not full WCAG conformance (W3C Accessibility Guidelines Working Group, 2023). Each report states whether a metric relied on source proxies, browser observations, or full action replay, so a weaker evidence configuration is never reported as the full protocol.

D.3 Evidence-Based Scoring and Aggregation

Scoring rules are fixed before evaluation. Each coverage score (components, states, views, tasks) is the percentage of applicable obligations with passing evidence: the component is usable, the state is reached, the view stays usable, or the action produces its expected outcome with the required context intact. Missing evidence earns nothing. Other checks use fixed rubrics. Applicability and rubric weights are set before execution, and empty requirement sets are marked not applicable.

When a necessary condition fails, a ceiling caps the affected metrics after raw scoring, and the lowest triggered cap applies. A startup failure sets every applicable dimension to 0. A failed required action caps task support and perceptual alignment at 40, and lost identity or value continuity caps data integrity, task support, and perceptual alignment at 40. An unreachable recovery state caps

state coverage, task support, and perceptual alignment at 50; a clipped required control caps view coverage and visual layout at 50; a failed required accessibility check caps accessibility at 50; and an unsafe action or provenance violation caps safety, licensing, and provenance at 40. Every applied ceiling is logged with its reason. A harness error is a measurement failure handled by the rerun policy, not a verdict on the product. The aggregate is the weighted mean of the capped scores over the applicable dimensions; all eleven weights are 1 in our experiments, and only inapplicable dimensions are excluded. A missing required observation gets no credit, whereas a test-harness failure is recorded as an incomplete measurement and handled by the declared rerun policy. Because a mean can hide trade-offs, we always report all eleven scores, blocking failures, completion rates, and evidence availability alongside it, and every comparison uses the same metric profile, weights, ceilings, and missing-evidence policy.

One defect, several dimensions. A single defect can lower several scores for different reasons. A clipped restore button violates visual layout, component usability, and task support; restoring the wrong value additionally violates data integrity. Reports therefore give each finding one identifier, the obligation it affects, and the evidence behind each score or ceiling it touches. This keeps correlated changes across dimensions from being read as independent confirmations of a gain.

D.4 Judge Configuration and Reporting

Every comparison fixes the judge model, prompts, decoding settings, evidence inputs, browser version, viewports, and scoring rules. The judge is Gemini-3.1-Pro at temperature 0 with 4,096 output tokens. Each artifact receives three semantic judgments; the median raw score is taken before the deterministic ceilings are applied. Harness faults are rerun twice and remain flagged if unresolved. The perceptual judge is always capped by the execution audit, and any semantic judge used for another dimension is reported with its combination rule. Reports keep raw and final scores, passed checks, localized failures, ceilings, browser observations, state probes, action results, and runtime errors. Perceptual records also keep the recordings, their description, the raw judgment, and the audit. Missing generations, completed outputs, and evaluation failures are reported separately, per family and per metric.

D.5 Scope

SAGE-Eval measures observable properties of the specified experience. Source heuristics cannot establish data correctness or standards conformance, and a finite set of browser probes cannot establish that every journey completes. Whether the scores track human judgment, whether each metric catches the defect it targets without moving unrelated scores, and how sensitive rankings are to the weights are tested in Appendix E. Long-term satisfaction, learnability, and trust would require dedicated user studies and are not measured here.

E Validating SAGE-Eval: Human Study and Controlled Defects

Perceptual alignment, visual layout, and data integrity are metrics we propose, so we checked them against expert ratings of the same aspects on the same interfaces.

Setup. Twenty experts took part (Table E.1). We sampled 88 tasks, 8 per family (2 low, 4 medium, and 2 high complexity), before looking at any scores. For each task, raters compared two outputs from the same model and budget: the no-memory generator and the full system. Each pair was rated independently by one HCI or frontend expert and one AI expert, blind to method and scores, with task order and left/right position randomized. Disagreements on the preferred interface were settled by a third blinded rater. The raters are experts rather than general end users, and long-term satisfaction is not measured.

Table E.1: **Human-study composition and assessment workload.** One pair assessment includes both interfaces for one task; extra adjudication is recorded separately.

Expertise	People	Pairs per evaluator	Pair assessments
HCI	6	4 people × 9; 2 × 8	52
Frontend engineering	4	4 people × 9	36
AI	10	8 people × 9; 2 × 8	88
Total	**20**	8–9 pairs per person	176

What raters scored. After using an interface on desktop and phone, raters answered one 7-point question per proposed metric, mirroring what that metric measures, and one overall question (Table E.2). They also marked each required action as pass or fail and chose the better interface.

Table E.2: **Human ratings against SAGE-EVAL scores.** Each question uses a 7-point scale and is paired with its corresponding automated metric. Arrows indicate no-memory → full-system means for the same 88 tasks (176 interfaces). Spearman ρ compares interface-level mean human ratings with automated scores.

Question asked to raters	SAGE-EVAL metric	Spearman ρ	Human (1–7)	SAGE-EVAL (0–100)
Is it clear what to do, and what happened after each action?	Perceptual alignment	0.76	4.75 → 5.12	70.20 → 72.10
Is content readable and every control visible in each view?	Visual layout	0.71	4.93 → 5.18	72.30 → 74.20
Do values and selections stay correct after edits and errors?	Data integrity	0.79	4.32 → 4.89	63.70 → 67.30
Overall, how well can you complete the task?	Aggregate score	0.74	4.81 → 5.19	72.10 → 74.20

Agreement with human ratings. SAGE-EVAL scores track human ratings well. The correlation (Spearman ρ) is 0.76 for perceptual alignment, 0.71 for visual layout, 0.79 for data integrity, and 0.74 between the aggregate score and overall usability. Humans and SAGE-EVAL also agree on the effect of memory: all three metrics improve from the no-memory to the full system on both scales, and data integrity improves most on both (Table E.2).

Raters agree reasonably with each other (Krippendorff's $\alpha = 0.67$ on overall usability). SAGE-EVAL picks the same better interface as the humans in 69 of 88 pairs (78.4%). Humans prefer the full system in 57 pairs, the no-memory system in 19, and rate 12 as ties. All required actions work in 74 of 88 full-system interfaces (84.1%), compared with 66 of 88 no-memory interfaces (75.0%).