ICONEDIT: BENCHMARKING AND IMPROVING PROGRESSIVE SVG EDITING WITH CHANGE-AND-PRESERVE REASONING

Anonymous authors
Paper under double-blind review

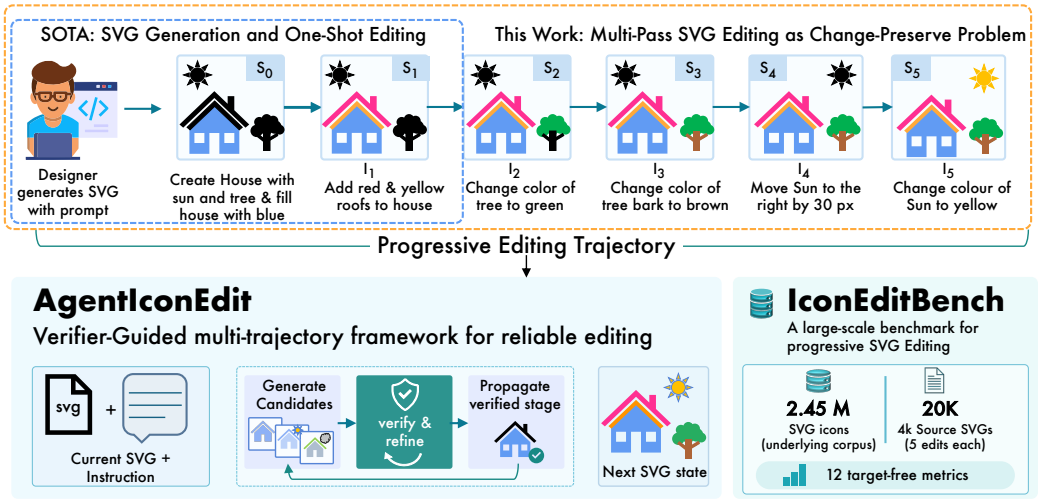

Figure 1: **Progressive SVG editing and our framework.** Each instruction I_i transforms S_{i-1} into S_i while preserving prior properties. AGENTICONEDIT verifies and refines candidate edits before state propagation, and ICONEDITBENCH enables large-scale target-free trajectory evaluation.

ABSTRACT

Progressive SVG icon editing requires models to execute each requested change while preserving visual and structural properties established by previous edits. Yet existing SVG editing benchmarks primarily evaluate isolated edits rather than sequential editing, where each generated output becomes the input to the next instruction. This distinction matters because an incomplete or unintended modification can propagate through subsequent edits, producing cumulative drift or degrading previously established content. We formulate progressive SVG editing as a *trajectory-level change-and-preserve* problem and introduce **IconEditBench**, an underlying corpus of 2.45 million SVG icons paired with edit instructions, together with a controlled benchmark of 4,000 source SVGs undergoing five sequential edits each. We further introduce twelve **target-free** metrics for measuring edit completion, preservation, drift, locality, validity, and trajectory-level reliability. We propose **AgentIconEdit**, a verifier-guided multi-trajectory framework that verifies and refines candidate edits before propagating them to subsequent states. Across four SOTA closed-source models, AgentIconEdit preserves perfect SVG validity while improving mean Edit Coverage by 4.8%, Spatial Placement by 31.6%, reducing geometric and topological drift by 37.5% and 35.3%, respectively. A 25-participant human study preferred AgentIconEdit over baseline in 80.8% of 25k pairwise comparisons. Together, these contributions provide a benchmark, evaluation metrics and a framework for studying and improving reliable progressive SVG editing. Code and dataset will be made publicly available.

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.

1 INTRODUCTION

Icons are fundamental visual elements in digital interfaces, communicating actions, objects, and concepts while contributing to visual consistency (Sun et al., 2026b). In practice, icon design is often iterative rather than one-shot: designers recolor components, modify geometry or style, add or remove elements, and refine earlier edits (Peng et al., 2026; Sun et al., 2026a; Kim et al., 2025a; Li et al., 2025). When these edits are applied sequentially, each modified asset becomes the starting point for the next edit. A reliable editor must therefore not only perform the requested change, but also preserve visual and structural properties established by previous edits. Scalable Vector Graphics (SVG) provide a natural representation for studying this problem. Unlike raster images, SVGs explicitly encode paths, primitives, groups, transforms, and styles, enabling fine-grained modification and direct inspection of intermediate states (Rodriguez et al., 2025; Xing et al., 2025; Wang et al., 2026a). This makes it possible to identify both intended and unintended modifications as an asset evolves. We therefore study *progressive SVG icon editing*, where a sequence of natural-language instructions is applied to an SVG, each generated output becoming the input to the next edit.

Recent works have advanced SVG generation and instruction-based editing (Wu et al., 2023; Rodriguez et al., 2025; Xing et al., 2025; Yang et al., 2026; Nishina & Matsui, 2024; 2025; Chen et al., 2025; Kuchar et al., 2026; Rodriguez et al., 2026; Gupta & Hebbar, 2026; Wang et al., 2026a). However, three limitations remain for progressive editing. *First*, existing benchmarks primarily evaluate independent edits rather than stateful sequences in which model outputs are propagated to subsequent steps. *Second*, existing metrics provide limited coverage of the *change-and-preserve* requirement: completing the requested edit while preserving unrelated content. *Third*, they do not characterize trajectory-level behavior such as cumulative drift, degradation of previously established content, or error propagation. Table 1 summarizes these differences.

To address these limitations, we formulate progressive SVG editing as a ***trajectory-level change-and-preserve problem***: at each step, an editor must execute the requested modification while preserving relevant properties accumulated from previous states. We introduce **IconEditBench**, a large-scale resource containing 2.45 million SVG icons paired with editing instructions. For controlled evaluation, we construct a balanced benchmark of 4,000 source SVGs drawn from MMSVG-Icon, MMSVG-Illustration, Text2SVG-Stack and SArena (Yang et al., 2026; Rodriguez et al., 2025; Wang et al., 2026a). Each source undergoes five sequential edits, yielding 20,000 progressive editing instances per evaluated system. Edit sequences cover color, geometry, style, opacity, outlines and element addition/removal, allowing editing to be studied throughout a trajectory rather in isolation.

Progressive editing requires evaluating both the requested change and preservation of unrelated and previously established content (Kim et al., 2025b; Ryu et al., 2025). Since multiple SVG realizations can satisfy an instruction, trajectories may legitimately diverge after the first edit, making a single fixed reference increasingly restrictive. We therefore introduce twelve target-free measures spanning edit execution, preservation and locality, validity, and trajectory-level reliability. They assess edit completion, content preservation, and error accumulation across the trajectory. Where possible, measures are computed deterministically from SVG structure or rendering, while semantic Edit Coverage is scored against instruction-specific requirements fixed before evaluation. This separation distinguishes failed edits from successful edits that unnecessarily alter unrelated content.

Building on this benchmark and evaluation protocol, we propose **AgentIconEdit**, a verifier-guided multi-trajectory framework for progressive SVG editing. It generates multiple candidates, verifies their structural and semantic correctness, and propagates the best state, using targeted refinement for unresolved requirements and best-so-far retention across iterations. This verification-before-propagation design limits the accumulation of incomplete or destructive edits. Across four closed-source models, AgentIconEdit maintains perfect SVG validity while improving mean Edit Coverage by 4.8%, Spatial Placement by 31.6%, and Structural Preservation by 1.3%, and reducing Excess Edit Distance by 9.6% relative to direct prompt editing (relative improvements averaged across models). These results demonstrate improved edit completion and localization while preserving structure across progressive editing trajectories.

In summary, our contributions are threefold: **(i)** we formulate *progressive SVG icon editing* as a stateful change-and-preserve problem and introduce *IconEditBench*, with a 2.45M SVG–instruction corpus and 4,000 SVGs undergoing five sequential edits each; **(ii)** we introduce twelve *target-free, trajectory-aware metrics* covering edit execution, preservation and locality, validity, and trajectory-

Approach	Instruction Editing	Stateful Sequential Editing	Preservation-Aware Metrics	Trajectory-Level Metrics	Verifier-Guided Framework	Data / Evaluation Scale
Earlier SVG generation and editing approaches						
IconShop (Wu et al., 2023)	Limited	✗	✗	✗	✗	–
SVGEditBench (Nishina & Matsui, 2024)	✓	✗	✗	✗	✗	1.3K
StarVector (Rodriguez et al., 2025)	–	✗	✗	✗	✗	–
LLM4SVG (Xing et al., 2025)	–	✗	✗	✗	✗	–
OmniSVG (Yang et al., 2026)	–	✗	✗	✗	✗	–
InternSVG (Wang et al., 2026a)	✓	✗	✗	✗	✗	31K
Recent SVG editing benchmarks						
SVGEditBench V2 (Nishina & Matsui, 2025)	✓	✗	✗	✗	✗	1.7K
SVGenius (Chen et al., 2025)	✓	✗	✗	✗	✗	2.4K
VectorEdits (Kuchar et al., 2026)	✓	✗	✗	✗	✗	270K
VectorGym (Rodriguez et al., 2026)	✓	✗	✗	✗	✗	6.5K
Vector-Bench (Gupta & Hebbar, 2026)	✓	✗	✓	✗	✗	40
IconEditBench + AgentIconEdit (ours)	✓	✓	✓	✓	✓	**2.45M**

Table 1: **Comparison with prior SVG generation and editing work.** We compare sequential editing, preservation and trajectory aware evaluation, verifier guided editing, and benchmark scale. IconEditBench contains 2.45M SVG instruction pairs and 20K progressive editing instances.

level reliability without requiring ground-truth edited SVGs; and **(iii)** we propose *AgentIconEdit*, a verifier-guided framework that verifies and refines candidate states before propagation, improving Edit Coverage, Spatial Placement and Structural Preservation while reducing Excess Edit Distance.

2 Related Work

SVG Editing. SVG editing remains less explored than generation. Text-Guided Vector Graphics Customization (Zhang et al., 2023) uses language-guided edits while preserving structure, while SVGEditBench (Nishina & Matsui, 2024) evaluates deterministic transformations. SVGEditBench V2 (Nishina & Matsui, 2025) and VectorEdits (Kuchar et al., 2026) extend to natural-language editing with source–target SVGs, and SVGenius (Chen et al., 2025) and InternSVG (Wang et al., 2026a) integrate editing into multimodal frameworks. VectorGym (Rodriguez et al., 2026) evaluates complex multi-operation edits, while Vector-Bench (Gupta & Hebbar, 2026) measures preservation of protected content during repair. Unlike these single-episode settings, we study *progressive SVG icon editing*, where outputs feed subsequent edits, exposing accumulated drift, regression of prior edits and propagation of unintended changes.

Benchmarks for SVG Editing. Existing benchmarks evaluate independent SVG editing episodes. SVGEditBench (Nishina & Matsui, 2024) covers six controlled transformations, SVGEditBench V2 (Nishina & Matsui, 2025) 1,683 source–target triplets, VectorEdits (Kuchar et al., 2026) over 270K edit pairs, and VectorGym (Rodriguez et al., 2026) complex expert-authored edits; SVGenius (Chen et al., 2025) and SArena (Wang et al., 2026a) additionally evaluate editing alongside understanding and generation. None evaluates stateful editing trajectories. From our 2.45M SVG–instruction corpus, we construct 4,000 source SVGs with five consecutive edits each (20,000 instances per system), propagating each generated state to evaluate both current edit completion and preservation of previously established content.

Evaluation of SVG Editing. Existing evaluation uses raster similarity (Nishina & Matsui, 2024), semantic and contour similarity (Nishina & Matsui, 2025), perceptual and semantic measures (Kuchar et al., 2026), preservation-aware checks (Gupta & Hebbar, 2026), and structural properties such as purity, coverage, compactness, and locality (Zhu et al., 2026). However, these methods do not capture how edit quality evolves across generated states. We introduce target-free measures of edit success, validity, preservation, drift, locality, integrity, and trajectory-level reliability. This trajectory-level formulation also motivates *AgentIconEdit*, which verifies edit completion and preservation before propagating each state.

3 IconEditBench

We introduce **IconEditBench**, a large-scale resource for progressive SVG editing, constructed from 2.45M source SVGs drawn from SArena (Wang et al., 2026a), MMSVG-Icon and MMSVG-Illustration (Yang et al., 2026), and Text2SVG-Stack (Rodriguez et al., 2025). Unlike existing benchmarks focused primarily on single-step editing (Wang et al., 2026a; Nishina & Matsui, 2025), *IconEditBench* considers edit sequences in which each instruction operates on the preceding state, requiring models to execute the current edit while preserving previously introduced changes.

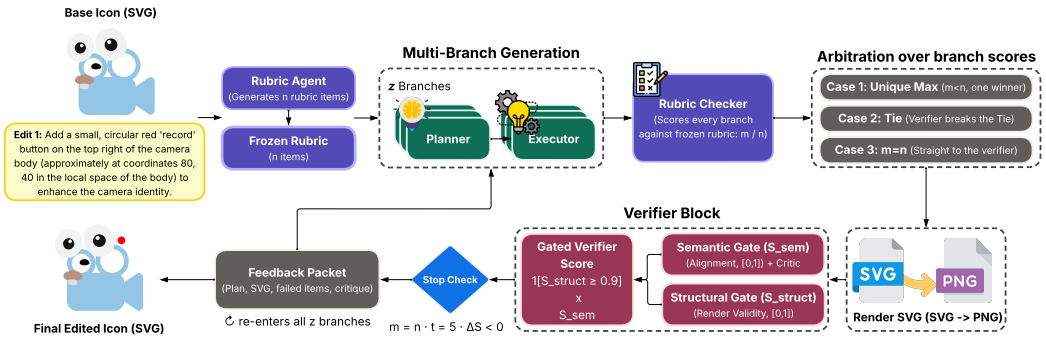

Figure 2: **Overview of AgentIconEdit.** A frozen edit rubric guides parallel planner-executor branches that generate and score candidate SVG edits. The Verifier Block evaluates semantic satisfaction and structural validity, while unresolved requirements and critiques are fed back for targeted refinement. The best candidate is retained across iterations until the stopping criteria are met, after which the final verified SVG is returned.

3.1 Benchmark Construction

Progressive Edit Chain Generation. For each source SVG S_0, we use `Gemma-4-31B-it` (Team et al., 2026) to construct a five-step trajectory, $S_0 \xrightarrow{I_1} S_1 \xrightarrow{I_2} \cdots \xrightarrow{I_5} S_5$. Given the minified SVG code, an available source caption, and admissible edit types, the model jointly plans exactly five structured instructions using schema-constrained decoding, without access to a rasterized image. Each I_t is defined over the state resulting from $I_{1:t-1}$, and a separate apply stage sequentially executes the instructions so that S_t is generated from S_{t-1}. Each edit must enrich the graphic, preserve its depicted subject, and avoid undoing or repeating prior modifications. The resulting $S_{1:5}$ are retained only as silver intermediate states for trajectory construction and are not used as ground-truth evaluation targets.

Edit Taxonomy. We define 14 operations across four families: *color*, *style*, *geometry*, and *semantic*. The first ten follow SArena's low-level categories (Wang et al., 2026a), covering color modification, opacity, outlining, style transformation, rotation, scaling, translation, flipping, and cropping. We add four higher-level operations—adding, removing, and replacing elements, and global theme restyling—that require reasoning about object identity, composition, and overall appearance and are common in instruction-guided image editing (Sheynin et al., 2024; Jia et al., 2026). Together, these operations cover both low-level SVG manipulations and semantic content changes. The complete taxonomy is provided to the planner, with constrained decoding ensuring valid operation types; diversity is encouraged within each chain, although repetitions are permitted.

Quality Control. We apply quality control at three stages. First, source SVGs are filtered for XML well-formedness, unsafe content, file size, and structural complexity. During chain generation, schema-constrained decoding enforces exactly five instructions from the predefined taxonomy. After each edit, the SVG is re-validated for XML well-formedness, structural validity, renderability, and safety, and failing trajectories are discarded. These checks establish syntactic and structural validity rather than semantic correctness, consistent with prior work separating technical validity from perceptual or semantic quality (Rodriguez et al., 2026; Wang et al., 2026b; Cipriano et al., 2026). Semantic compliance is evaluated separately using our rubric-based evaluation, whose agreement with human judgments is examined in our human evaluation study.

4 AgentIconEdit

Given the current SVG state S_{t-1} and instruction I_t, **AgentIconEdit** first constructs a fixed edit rubric, explores multiple editing strategies through parallel planner–executor branches, and verifies the resulting candidates to select or refine the best result. As shown in Figure 2, the framework comprises a *Rubric Agent*, *MultiBatch Generation*, and a *Verifier Block*. Generation and verification repeat as needed while retaining the strongest candidate across refinement rounds.

4.1 RUBRIC AGENT

Before candidate generation, the Rubric Agent constructs a fixed verification checklist defining the required changes and preservation constraints. Using the instruction I_t, SVG source S_{t-1}, and properties established by previous edits $\mathcal{P}_{<t}$, it reasons directly over SVG code without a rendered image and produces atomic, independently verifiable criteria. Each specifies a required change, e.g., adding or removing an element, changing color or geometry, or adjusting position and whether it should be verified from the rendering or SVG source.

Preservation criteria are appended deterministically rather than model-generated: one requires untargeted SVG content to remain unchanged, and one is added for every property established by prior edits. Formally, $\mathcal{R}_t = G(I_t, S_{t-1}, \mathcal{P}_{<t}) \cup \{c_{\text{pres}}\} \cup \{c(p) \mid p \in \mathcal{P}_{<t}\}$,, where $G(\cdot)$ contains instruction-derived criteria, c_{pres} preserves unrelated content, and $c(p)$ preserves an established property p. The instruction-derived portion typically contains three to six criteria and is capped at 12. The complete rubric $\mathcal{R}_t$ is generated once and frozen across all candidate and refinement rounds, ensuring consistent verification. Failed criteria identify unresolved changes or preservation violations, while candidates satisfying more criteria provide stronger starting points for refinement.

4.2 MULTI-BATCH GENERATION

Because an SVG edit may admit multiple valid code-level realizations, AgentIconEdit uses a best-of-N strategy rather than relying on a single generation. Each batch contains $z(= 3)$ independent planner–executor branches initialized from the same S_{t-1} and evaluated against the frozen rubric $\mathcal{R}_t$. We study the effect of z in Section 6.2.

Planner Agents. Each planner receives S_{t-1}, I_t, $\mathcal{R}_t$, and a deterministic SVG inventory containing available identifiers, classes, element counts, and geometric and color hints, reducing references to nonexistent or ambiguous targets. Each branch independently samples a structured plan P_j with one or more operations specifying the action, target, concrete attribute or geometric changes, insertion position when needed, targeted rubric criteria, and elements to preserve. Branches share the same inputs, prompting procedure, and objective, differing only through independent sampling.

Executor Agents. Each plan is executed against the original SVG and frozen rubric: $\hat{S}_t^{(j)} = E(S_{t-1}, P_j, \mathcal{R}_t), \quad j \in \{1, \ldots, z\}$.. The executor applies operations sequentially and returns a complete SVG rather than a patch. Marked elements remain unchanged, while canvas properties, referenced definitions, and SVG structure are preserved whenever possible. Each candidate undergoes structural preflight; detected failures trigger one targeted repair attempt rather than full regeneration. The z candidates are then passed to the Verifier Block.

4.3 VERIFICATION AND REFINEMENT

The Verifier Block evaluates whether each candidate satisfies the requested edit while remaining structurally faithful to the input. All candidates and refinement rounds use same frozen rubric $\mathcal{R}_t$.

Rubric Verification. For candidate $\hat{S}_t^{(j)}$, each criterion in $\mathcal{R}_t = \{c_1, \ldots, c_K\}$ receives a binary outcome $r_k^{(j)} \in \{0, 1\}$, yielding: $S_{\text{sem}}^{(j)} = \frac{1}{K} \sum_{k=1}^{K} r_k^{(j)}$. The verifier also records unsatisfied criteria and a critique for targeted refinement. Since the rubric is frozen before generation, all candidates are evaluated against identical requirements.

Structural Verification. Rubric satisfaction alone may conceal unnecessary rewriting or structural damage. Candidates therefore undergo hard validity checks for SVG parsing, a valid root, safe content, non-blank rendering, and preservation of the original viewBox. Surviving candidates receive a structural consistency score $S_{\text{struct}}^{(j)} \in [0, 1]$, combined with semantic verification as: $S_{\text{final}}^{(j)} = \mathbb{1}\left[S_{\text{struct}}^{(j)} \geq 0.9\right] S_{\text{sem}}^{(j)}$.. Thus, semantic satisfaction contributes only when the candidate passes structural gate, preventing structurally damaging edits from receiving high final scores.

Selection and Refinement. A candidate satisfying the complete rubric and structural gate is accepted. Otherwise, the candidate with the highest rubric completion score is selected, with ties resolved by the Verifier Block. Its plan, SVG, unsatisfied criteria and verifier feedback are shared with all planner branches for the next MultiBatch round. Crucially, every refinement remains rooted

at the original S_{t-1}: the best candidate serves only as a reference, not the new editing base. This avoids compounding candidate-level errors and lets each branch generate a corrected edit from the clean input. AgentIconEdit retains the best verified candidate across rounds, replacing it only upon score improvement. Refinement stops upon complete rubric and structural satisfaction, score degradation or five rounds. If no candidate fully satisfies the requirements, the best verified candidate observed across all rounds is returned.

5 METRICS FOR SVG ICON EDITING

SVG editing evaluation must capture not only rendered similarity, but also instruction satisfaction, SVG usability, preservation, and stability across successive edits. We define 12 target-free metrics spanning *edit correctness and validity*, *preservation and locality*, and *progressive editing* (Brooks et al., 2023; Chen et al., 2025; Wu et al., 2023). They operate on the instruction, pre- and post-edit SVG states, and their renders, without requiring ground-truth edited SVGs. Appendix 9 summarizes all metrics, evaluation levels, and optimization directions.

5.1 EDIT CORRECTNESS AND VALIDITY

SVG Validity. We evaluate validity at three levels. L1 requires a parseable SVG with a valid root and at least one drawable element. L2 additionally requires non-blank rendering and all drawable geometry within the viewBox. L3 further requires all internal references to resolve. A separate non-degeneracy check (ND) requires the rendered ink ratio relative to the previous state to lie within $[0.5, 3.0]$. All four indicators are reported separately.

Edit Coverage. Edit Coverage (EC) measures execution of the requested transformation. Before evaluating any system, each instruction is decomposed into a frozen set of atomic requirements $\mathcal{R}_t = r_1, \ldots, r_K$. A vision-language judge compares pre- and post-edit renders and evaluates each requirement: $\mathrm{EC}_t = \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}[r_k \text{ is satisfied}]$.. The same requirements are used across systems, preventing evaluation targets from adapting to individual outputs.

Colour Adherence. For colour-specific instructions, Colour Adherence (CA) measures whether the requested colour appears in the edited SVG. Let c^* be the requested colour and d the minimum CIEDE2000 distance to any resolved fill. We report $\mathrm{CA} * h = \mathbb{1}[d \leq 10]$ and $\mathrm{CA} * s = \exp(-d^2/2\sigma^2)$, with $\sigma = 15$.

Spatial Placement. Spatial Placement (PL) evaluates instructions specifying explicit coordinates. Newly introduced elements are identified from the SVG difference. The hard score tests if each requested coordinate lies within a new element's bounding box; soft score decreases with normalized distance to nearest new-element centroid. Instructions without explicit positions are excluded.

Successful Edit Score. Successful Edit Score (SES) indicates whether an edit produces a structurally usable, non-degenerate SVG: $\mathrm{SES}_t = \mathbb{1}[\mathrm{L1} \wedge \mathrm{L2} \wedge \mathrm{L3} \wedge \mathrm{ND}]$.. SES measures output usability, while semantic instruction satisfaction is captured separately by EC.

5.2 PRESERVATION AND EDIT LOCALITY

Structural metrics use canonicalized SVGs with transforms and inherited presentation attributes resolved before comparison. Consecutive states are aligned using Hungarian assignment with cost: $C(e, e') = 0.4(1 - \mathrm{idSim}(e, e')) + 0.4(1 - \mathrm{IoU}(e, e')) + 0.2\frac{\Delta E_{00}(e,e')}{100}$,, with pairs satisfying $C(e, e') > 0.6$ left unmatched. If both elements lack identifiers, the identifier term is omitted and its weight redistributed.

Structural Preservation. Structural Preservation (SP) captures element survival, attribute preservation, and paint-order preservation. For matching M, $\mathrm{SP}_{\mathrm{set}} = \frac{|\{e \in G_{t-1} : M(e) \neq \emptyset\}|}{|G_{t-1}|}$, $\mathrm{SP}_{\mathrm{attr}} = \frac{|\{e : \mathrm{attrs}(e) = \mathrm{attrs}(M(e))\}|}{|\{e : M(e) \neq \emptyset\}|}$,, while $\mathrm{SP}_{\mathrm{ord}}$ is Kendall's τ_b between matched-element paint orders before and after editing. The three components are reported separately. Without target-element annotations, SP measures overall preservation and may include intentional changes.

Target Integrity. Target Integrity (TI) measures whether modified elements change only in instruction-licensed ways. Instruction terms such as *colour*, *move*, *scale*, and *outline* are deterministically mapped to permitted SVG attributes. TI is the fraction of changed matched elements whose modified attributes fall entirely within this set, distinguishing targeted edits from unnecessary changes to the same element.

Excess Edit Distance. Excess Edit Distance (EED) quantifies structural changes through deleted and inserted elements, tag changes, and modified attributes: $\mathrm{EED}_t = \frac{N_{\mathrm{del}}+N_{\mathrm{ins}}+N_{\mathrm{tag}}+N_{\mathrm{attr}}}{|G_{t-1}|}$.. Lower values mean more surgical edits. Without target annotations, EED covers complete document and provides an upper bound on unnecessary structural change rather than a fully scope-aware distance.
Style Consistency. Style Consistency (SC) measures whether new elements match the pre-edit icon. A style profile from G_{t-1} captures stroke widths, color palette, coordinate grid, and path-complexity distribution. SC is the fraction of applicable profile checks passed by new elements; palette consistency is omitted when a new color is explicitly requested.

5.3 Progressive Editing Metrics

Cumulative Drift. To capture changes accumulating across steps, we compare the original SVG G_0 with the final available state G_T. For elements surviving the trajectory, geometric drift is: $\mathrm{CD}_{\mathrm{geo}} = \frac{1}{|B|}\sum_{e\in B}\frac{\|c(e)-c(M(e))\|_2}{\sqrt{w^2+h^2}}$,, where $c(\cdot)$ denotes the centroid and (w, h) the original viewBox dimensions. Colour drift is: $\mathrm{CD}_{\mathrm{col}} = \frac{1}{|B|}\sum_{e\in B}\Delta E_{00}\big(\mathrm{fill}(e), \mathrm{fill}(M(e))\big)$,, while $\mathrm{CD}_{\mathrm{top}}$ is the fraction of surviving elements whose relative paint order changes. Lower values mean less drift. As target annotations are unavailable, these measures may also capture intentional transformations.
Trajectory. We treat structural failure as a chain-termination event, with a step remaining alive when it satisfies L2 and ND. The survival curve is: $S(t) = \frac{\#\{\text{chains alive at step } t\}}{\#\{\text{chains reaching step } t\}}$, and Expected Chain Length is: $\mathrm{ECL} = \sum_{t=1}^{5} S(t)$,, with a maximum of five. We also fit the slope β_{SP} of $\mathrm{SP}_{\mathrm{attr}}$ against edit position to measure progressive structural degradation.
Failure Mode Distribution. Failure Mode Distribution (FMD) explains why edits fail by assigning each failed step a primary diagnostic label under a fixed precedence order. Categories include truncated outputs, no-ops, off-canvas geometry, render failures, degeneration, collateral deletion or mutation, colour errors, z-order changes, and style breaks. Reported as a distribution rather than a scalar, FMD complements the quantitative metrics by identifying dominant failure sources.

6 Experiments

Models. We evaluate four proprietary models—`GPT-5.4-mini` (OpenAI, 2026a), `GPT-5.5` (OpenAI, 2026b), `Claude-Sonnet-4-6` (Anthropic, 2026b), and `Claude-Opus-4-6` (Anthropic, 2026a)—and two open-source models, `Qwen3-8B` (Yang et al., 2025) and `Qwen3.5-9B` (Qwen Team, 2026), using provider APIs and local inference, respectively. Each is tested in *Base* (direct editing) and *Agentic* (AgentIconEdit) settings. In AgentIconEdit, the same model handles rubric generation/checking, planning, and execution, with `gemini-3.1-flash-image` (Google DeepMind, 2026) as a fixed multimodal judge.
Evaluation Protocol. We evaluate 4K SVGs (1K each from SArena (Wang et al., 2026a), MMSVG-Icon, MMSVG-Illustration (Yang et al., 2026) and Text2SVG-Stack (Rodriguez et al., 2025)), sampled with a fixed seed and stored IDs. Each has a predefined 5-edit chain generated by `gemma4-31B-it` (Gemma Team, 2026), yielding 20K instances per model/method. Edits are sequentially applied, with identical sources, instructions, ordering across settings.
Baseline. At each step, Base directly generates a complete SVG from S_{t-1} and I_t, without planning, verification, branching, or refinement; each valid output is propagated forward.
Implementation Details. AgentIconEdit uses $z = 3$ planner–executor branches, a frozen rubric, $S_{\mathrm{struct}} \geq 0.9$, best-so-far retention, and up to five refinement rounds per edit. SVG generation uses temperature 0.3 unless noted, with provider-default remaining parameters. Inputs are unnormalized; outputs only undergo complete-SVG extraction, without XML repair. CairoSVG (Ayoub & CourtBouillon contributors, 2026) renders at 512×512. API timeouts are 120,s, except GPT-5.5 and Claude Opus 4.6 Base (420,s); inference is API-based; rendering and metrics use CPU nodes.
Evaluation Metrics. IconEditBench (Section 5) measures SVG validity, structural preservation, geometric/colour/topological drift, failing steps, trajectory survival, Edit Coverage, Colour Adherence, Spatial Placement, Target Integrity, Excess Edit Distance, Style Consistency, and successful-edit rate. Edit Coverage uses fixed judge `GPT-5.6-terra` (OpenAI, 2026c); metrics use their applicable progressive-chain instances rather than a shared denominator.

Model	Reliability		Edit Execution			Preservation & Locality				Trajectory				
	SVG Validity (↑)	SES (↑)	Edit Coverage (↑)	Colour Adh. (↑)	Spatial (↑)	Struct. Pres. (↑)	Target Integrity (↑)	Excess Edit Distance (↓)	Style (↑)	Geom. Drift (↓)	Colour Drift (↓)	Topo. Drift (↓)	Failing Steps (↓)	Survival (↑)
Open Source Models														
Qwen3-8B$_{\text{Base}}$	0.975	0.904	0.495	0.712	0.821	0.886	**0.891**	0.693	0.916	0.009	15.29	0.010	66	0.910
Qwen3-8B$_{\text{AgentIconEdit (Ours)}}$	**1.000**	**0.966**	**0.542**	**0.747**	**0.826**	**0.891**	0.880	**0.666**	**0.922**	**0.005**	**13.48**	**0.008**	**64**	**0.913**
$\Delta_{\text{Ours-Base}}$ (Qwen3-8B)	+2.6%	+6.9%	+9.5%	+4.9%	+0.6%	+0.6%	-1.2%	+3.9%	+0.7%	+44.4%	+11.8%	+20.0%	+3.0%	+0.3%
Qwen3.5-9B$_{\text{Base}}$	0.986	0.957	0.503	0.701	0.811	0.886	0.857	0.826	0.911	0.003	13.53	0.021	67	0.891
Qwen3.5-9B$_{\text{AgentIconEdit (Ours)}}$	**1.000**	**0.980**	**0.607**	**0.704**	**0.823**	**0.891**	**0.865**	**0.670**	**0.950**	0.003	**11.54**	**0.018**	**65**	**0.914**
$\Delta_{\text{Ours-Base}}$ (Qwen3.5-9B)	+1.4%	+2.4%	+20.7%	+0.4%	+1.5%	+0.6%	+0.9%	+18.9%	+4.3%	0.0%	+14.7%	+14.3%	+3.0%	+2.6%
Closed Source Models														
GPT-5.4-mini$_{\text{Base}}$	1.000	0.980	0.631	0.680	0.634	0.890	0.891	0.763	0.919	0.003	15.02	0.034	70	0.914
GPT-5.4-mini$_{\text{AgentIconEdit (Ours)}}$	1.000	0.980	**0.711**	**0.712**	**1.000**	**0.897**	**0.909**	**0.736**	**0.948**	0.003	**14.03**	**0.024**	**65**	0.914
$\Delta_{\text{Ours-Base}}$ (GPT-5.4-mini)	0.0%	0.0%	+12.7%	+4.7%	+57.7%	+0.8%	+2.0%	+3.5%	+3.2%	0.0%	+6.6%	+29.4%	+7.1%	0.0%
GPT-5.5$_{\text{Base}}$	1.000	0.980	0.749	0.708	0.644	0.891	0.870	0.772	0.866	0.005	14.68	0.105	71	0.914
GPT-5.5$_{\text{AgentIconEdit (Ours)}}$	1.000	0.980	**0.760**	**0.726**	**0.909**	**0.909**	**0.910**	**0.701**	**0.870**	**0.003**	**11.29**	**0.051**	**68**	0.914
$\Delta_{\text{Ours-Base}}$ (GPT-5.5)	0.0%	0.0%	+1.5%	+2.5%	+41.2%	+2.0%	+4.6%	+9.2%	+0.5%	+40.0%	+23.1%	+51.4%	+4.2%	0.0%
Opus 4.6$_{\text{Base}}$	1.000	0.980	0.711	0.685	0.641	0.885	0.866	0.847	**0.885**	0.010	18.95	0.105	**71**	0.914
Opus 4.6$_{\text{AgentIconEdit (Ours)}}$	1.000	0.980	**0.717**	**0.717**	**0.804**	**0.896**	**0.870**	**0.687**	0.876	**0.003**	**17.34**	**0.054**	72	0.914
$\Delta_{\text{Ours-Base}}$ (Opus 4.6)	0.0%	0.0%	+0.8%	+4.7%	+25.4%	+1.2%	+0.5%	+18.9%	-1.0%	+70.0%	+8.5%	+48.6%	-1.4%	0.0%
Sonnet 4.6$_{\text{Base}}$	1.000	0.980	0.714	0.692	0.694	0.889	0.803	0.760	0.915	0.005	13.81	0.059	72	0.914
Sonnet 4.6$_{\text{AgentIconEdit (Ours)}}$	1.000	0.980	**0.745**	**0.727**	**0.708**	**0.899**	**0.805**	**0.710**	**0.927**	**0.003**	**13.73**	**0.052**	**69**	0.914
$\Delta_{\text{Ours-Base}}$ (Sonnet 4.6)	0.0%	0.0%	+4.3%	+5.1%	+2.0%	+1.1%	+0.2%	+6.6%	+1.3%	+40.0%	+0.6%	+11.9%	+4.2%	0.0%

Table 2: Comparison of direct prompt editing (*Base*) and our approach (**AgentIconEdit**) across six models. Metrics are grouped by the property they evaluate. ↑ indicates higher is better and ↓ lower is better. Bold marks the better result within each model; shaded rows correspond to AgentIconEdit.

6.1 Quantitative and Qualitative Evaluation

Table 2 shows that *AgentIconEdit improves edit completion while reducing unnecessary changes*. Across six models, Edit Coverage rises from 0.634 to 0.680 (average per-model relative improvement of +8.3%), Spatial Placement from 0.708 to 0.845 (+21.4%), and Excess Edit Distance falls from 0.777 to 0.695 (−10.2%); Edit Coverage and Excess Edit Distance improve for every model. Trajectory stability also improves, with geometric, topological, and colour drift decreasing by 32.4%, 29.3%, and 10.9%, respectively; colour and topological drift improve for all models. These gains remain visible despite near-saturated SVG Validity, SES, and Survival for several baselines. Benefits differ by model family. For open-source models, mean Edit Coverage increases from 0.499 to 0.575 (+15.1%), SES from 0.931 to 0.973, and validity reaches 1.000. Closed-source models already have perfect validity and 0.980 SES, but improve Spatial Placement from 0.653 to 0.855 (+31.6%), with geometric and topological drift decreasing by 37.5% and 35.3%. Structural Preservation and colour and topological drift improve for all six models; geometric drift improves for four and is unchanged for two. Improvements are not universal: Qwen3-8B slightly decreases in Target Integrity, Opus 4.6 slightly decreases in Style Consistency and incurs one additional failing step. AgentIconEdit therefore substantially reduces cumulative drift without guaranteeing improvement in every preservation property, retaining model-specific differences and failure modes.

Human Studies. Human evaluation with 25 participants further validates both the generated rubrics and editing quality: 86.9% of rubric items were judged valid, with a mean completeness rating of 4.52/5. AgentIconEdit also improved instruction satisfaction (4.35 vs. 2.58), preservation (4.31 vs. 3.68), and target correctness (84.4% vs. 50.8%), and was preferred over Base in 80.8% of 25k pairwise comparisons. We provide more details in the Appendix 14.

Figure 3 compares representative five-step Base and **AgentIconEdit** trajectories. In the mountain example, Base incorrectly extends the requested smaller background peak into the sun (Edit 1), places snow on this new peak rather than the highest peak (Edit 3), and renders the subtle outline as a rectangular border (Edit 5); AgentIconEdit preserves the intended arrangement, correctly places the snow, and retains prior edits while adding the birds and final outline. In the graduation-cap example, Base fails to visibly add the top-left white highlight (Edit 2), produces a jumbled rather than three-strand tassel (Edit 3), and mispositions the gold seal relative to the tassel string (Edit 4), whereas AgentIconEdit better executes these changes while preserving the recolored cap and prior details. These examples show how early errors propagate through progressive trajectories and align

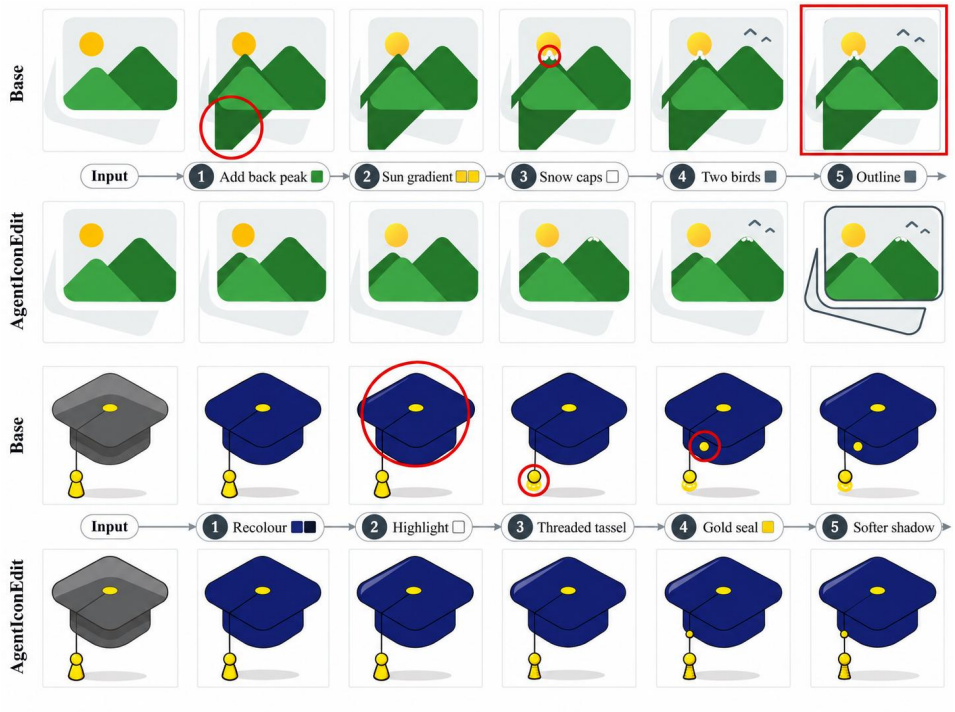

Figure 3: **Qualitative comparison of progressive SVG editing.** Red annotations highlight representative Base failures that persist across sequential edits, while AgentIconEdit better executes requested changes and preserves prior edits.

Verifier	EC ↑	SP ↑	EED ↓	TI ↑	GD ↓
✗	0.486	0.851	0.792	0.800	0.0039
✓	**0.760**	**0.909**	**0.701**	**0.910**	**0.0030**

Table 3: **Effect of verifier-guided editing.** With verifier we obtain substantial gains.

Rubric	EC ↑	SP ↑	EED ↓	TI ↑	GD ↓
✗	0.573	0.891	0.774	0.762	0.0035
✓	**0.760**	**0.909**	**0.701**	**0.910**	**0.0030**

Table 4: **Impact of the Rubric Agent.** Having a rubric agent adds to substantial gains.

with the improvements in edit execution, spatial placement, and preservation in Table 2; additional qualitative examples (and failure cases) appear in Appendix.

6.2 Ablation Studies

We used GPT-5.5 in the Tabs 3-4 and in Appendix. In Tab 3, we observe that the full AgentIconEdit framework improves EC by 56.4%, SP by 6.8%, TI by 13.8%, reduces EED by 11.5% and GD by 23.1%, as compared to w/o verification. Similarly, in Tab 4, the full AgentIconEdit framework improves EC by 32.6%, TI by 19.4%, SP by 2.0% reduces EED by 9.4%, GD by 14.3%, showing that rubric-based verification improves instruction execution, locality and trajectory stability.

7 Conclusion

We introduced *progressive SVG icon editing*, where sequential state propagation can accumulate edit and preservation errors. We presented *IconEditBench*, comprising a 2.45M SVG–instruction corpus and 4K icons with five sequential edits, alongside *target-free, trajectory-aware metrics* for edit correctness, preservation, and reliability. We further proposed *AgentIconEdit*, which verifies and refines candidate edits before propagation. Across six model backbones, AgentIconEdit improves Edit Coverage and Structural Preservation while reducing Excess Edit Distance, with some model-dependent trade-offs. Our results highlight preservation and trajectory-level reliability as key challenges beyond producing valid SVGs.

AI USE STATEMENT

Generative AI tools were used during manuscript preparation solely for language editing, including polishing phrasing, correcting spelling and grammar, and improving the clarity and readability of author-written text. All AI-assisted edits were reviewed and revised by the authors to ensure that they preserved the intended technical meaning, scientific claims, results, and conclusions. Generative AI was not used to independently determine the scientific contributions, formulate research claims, or interpret experimental results. The language models and vision–language models used as experimental systems, benchmark-generation components, or evaluators are part of the research methodology and are explicitly described in the corresponding sections of the paper. The authors take full responsibility for the final content of this work, including all text, claims, analyses, results, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work was conducted under an approved institutional research ethics protocol for studying the performance of LLM-assisted tools for specialized tasks; identifying information associated with the ethics approval is withheld during double-blind review. All human-participant procedures, where applicable, followed the approved protocol, including informed consent, voluntary participation, and appropriate handling of study data. The benchmark is constructed from publicly available SVG datasets and is used for research purposes in accordance with their respective terms and licenses. We do not intend the proposed methods or evaluations to enable harmful applications, and we report model limitations and failure modes to support transparent and responsible development of LLM-assisted design tools.

REPRODUCIBILITY STATEMENT

We have taken several steps to support reproducibility. Section 6 describes the benchmark construction, deterministic hash-based sampling procedure, evaluated models, progressive editing protocol, evaluation setup, and compute configuration. Section 5 specifies the complete metric suite, including all thresholds and validity checks used in our experiments. Each experimental report records the fixed sample identifiers and their hashes, the frozen instruction-requirement store and its hash, evaluator configuration, metric thresholds, and the Git commit corresponding to the scoring implementation. Additional implementation details, prompts, benchmark-generation procedures, evaluation code, and configuration files are provided in the supplementary material. We will also release the benchmark construction and evaluation code, together with the finalized benchmark metadata, to facilitate independent reproduction and extension of our experiments.

REFERENCES

Anthropic. Claude Opus 4.6 System Card, February 2026a. URL https://www.anthropic.com/claude-opus-4-6-system-card.

Anthropic. Claude Sonnet 4.6 System Card, February 2026b. URL https://www.anthropic.com/claude-sonnet-4-6-system-card.

Guillaume Ayoub and CourtBouillon contributors. CairoSVG: A simple svg converter based on cairo. Software, 2026. URL https://cairosvg.org/. Version 2.2.1.

Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18392–18402. IEEE, 2023.

Siqi Chen, Xinyu Dong, Haolei Xu, Xingyu Wu, Fei Tang, Hang Zhang, Yuchen Yan, Linjuan Wu, Wenqi Zhang, Guiyang Hou, Yongliang Shen, Weiming Lu, and Yueting Zhuang. Svgenius: Benchmarking llms in svg understanding, editing and generation. In *Proceedings of the 33rd ACM International Conference on Multimedia*, MM '25, pp. 13289–13296, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400720352. doi: 10.1145/3746027.3758287. URL https://doi.org/10.1145/3746027.3758287.

Marco Cipriano, Leonardo Zini, Alexandra Schild, Valentin Teutschbein, Afsana Mimi, Marcella Cornia, Lorenzo Baraldi, and Gerard de Melo. Svg-score: Human-aligned evaluation of text-to-svg generation. *arXiv preprint arXiv:2609.03806*, 2026.

Gemma Team. Gemma 4 Technical Report, 2026. URL https://arxiv.org/abs/2607.02770.

Google DeepMind. Gemini 3.1 Flash Image Model Card, February 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-flash-image/.

Y. Gupta and Prannay Hebbar. Vector-bench: Can models surgically edit svg code? *ArXiv*, abs/2607.19056, 2026. URL https://api.semanticscholar.org/CorpusID:290404017.

Bohan Jia, Wenxuan Huang, Yuntian Tang, Junbo Qiao, Jincheng Liao, Shaosheng Cao, Fei Zhao, Zhaopeng Feng, Zhouhong Gu, Zhenfei Yin, Lei Bai, Wanli Ouyang, Lin Chen, Fei Zhao, Zihan Wang, Yuan Xie, and Shaohui Lin. Compbench: Benchmarking complex instruction-guided image editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1112–1122, June 2026.

Daneul Kim, Jaeah Lee, and Jaesik Park. Improving editability in image generation with layer-wise memory. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7889–7898, 2025a. doi: 10.1109/CVPR52734.2025.00739.

Yoonjeon Kim, Soohyun Ryu, Yeonsung Jung, Hyunkoo Lee, Joowon Kim, June Yong Yang, Jaeryong Hwang, and Eunho Yang. Preserve or modify? context-aware evaluation for balancing preservation and modification in text-guided image editing. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23474–23483, 2025b. doi: 10.1109/CVPR52734.2025.02186.

Josef Kuchar, Marek Kadlcik, Michal Spiegel, and Michal Stefanik. VectorEdits: A dataset and benchmark for instruction-based editing of vector graphics. In Stelios Piperidis, Núria Bel, Henk van den Heuvel, Nancy Ide, Simon Krek, and Antonio Toral (eds.), *Proceedings of the Fifteenth Language Resources and Evaluation Conference*, pp. 11119–11124, Palma de Mallorca, Spain, May 2026. ELRA Language Resource Association. doi: 10.63317/5gc5ibtb5k8i. URL https://aclanthology.org/2026.lrec-1.868/.

Zhuoying Li, Zhu Xu, Yuxin Peng, and Yang Liu. Balancing preservation and modification: A region and semantic aware metric for instruction-based image editing. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 36576–36596. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/li25dh.html.

Kunato Nishina and Yusuke Matsui. Svgeditbench: A benchmark dataset for quantitative assessment of llm's svg editing capabilities. *arXiv preprint arXiv:2404.13710*, 2024.

Kunato Nishina and Yusuke Matsui. Svgeditbench v2: A benchmark for instruction-based svg editing. *arXiv preprint arXiv:2502.19453*, 2025.

OpenAI. GPT-5.4 Thinking System Card. OpenAI Deployment Safety Hub, March 2026a. URL https://deploymentsafety.openai.com/gpt-5-4-thinking.

OpenAI. GPT-5.5 System Card. OpenAI Deployment Safety Hub, April 2026b. URL https://deploymentsafety.openai.com/gpt-5-5.

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

Xiaohan Peng, Debanjana Haldar, Wendy E. Mackay, and Janin Koch. Designtrace: Exploring, iterating and tracking design alternatives with genai. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722783. doi: 10.1145/3772318.3791036. URL https://doi.org/10.1145/3772318.3791036.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images and text. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16175–16186, 2025. doi: 10.1109/CVPR52734.2025.01508.

Juan A. Rodriguez, Haotian Zhang, Abhay Puri, Aly Shariff, Meng lin, Xiaoqing Xie, Tianyang Zhang, Rishav Pramanik, Sai Rajeswar, Perouz Taslakian, Spandana Gella, David Vazquez, Christopher Pal, and Marco Pedersoli. Vectorgym: A multi-task benchmark for SVG code generation and manipulation, 2026. URL https://openreview.net/forum?id=DBFbNT65xO.

Suho Ryu, Kihyun Kim, Eugene Baek, Dongsoo Shin, and Joonseok Lee. Towards scalable human-aligned benchmark for text-guided image editing. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18292–18301, 2025. doi: 10.1109/CVPR52734.2025.01705.

Shelly Sheynin, Adam Polyak, Uriel Singer, Yuval Kirstain, Amit Zohar, Oron Ashual, Devi Parikh, and Yaniv Taigman. Emu edit: Precise image editing via recognition and generation tasks. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8871–8879, 2024. doi: 10.1109/CVPR52733.2024.00847.

Zhida Sun, Xiaodong Wang, Zhenyao Zhang, Min Lu, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Iconix: Controlling semantics and style in progressive icon grids generation. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, New York, NY, USA, 2026a. Association for Computing Machinery. ISBN 9798400722783. doi: 10.1145/3772318.3790694. URL https://doi.org/10.1145/3772318.3790694.

Zhida Sun, Xiaodong Wang, Zhenyao Zhang, Min Lu, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Iconix: Controlling semantics and style in progressive icon grids generation. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, New York, NY, USA, 2026b. Association for Computing Machinery. ISBN 9798400722783. doi: 10.1145/3772318.3790694. URL https://doi.org/10.1145/3772318.3790694.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Haomin Wang, Jinhui Yin, Qi Wei, Wenguang Zeng, Lixin Gu, Shenglong Ye, Zhangwei Gao, Yaohui Wang, Yanting Zhang, Yuanqi Li, et al. Internsvg: Towards unified svg tasks with multimodal large language models. In *International Conference on Learning Representations*, volume 2026, pp. 38649–38705, 2026a.

Yiming Wang, Ye Chen, Hanqi Chen, and Bingbing Ni. Svgeval: A vision-grounded framework for perceptual-quality benchmarking and evaluation in text-to-svg generation. *arXiv preprint arXiv:2608.01977*, 2026b.

Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. Iconshop: Text-guided vector icon synthesis with autoregressive transformers. *ACM Trans. Graph.*, 42(6), December 2023. ISSN 0730-0301. doi: 10.1145/3618364. URL https://doi.org/10.1145/3618364.

Ximing Xing, Juncheng Hu, Guotao Liang, Jing Zhang, Dong Xu, and Qian Yu. Empowering llms to understand and generate complex vector graphics. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19487–19497, 2025. doi: 10.1109/CVPR52734.2025.01815.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Fukun Yin, Jiaxu Zhang, Liao Wang, Gang Yu, Xingjun Ma, and Yu-Gang Jiang. Omnisvg: A unified scalable vector graphics generation model. *Advances in Neural Information Processing Systems*, 38:113670–113696, 2026.

Peiying Zhang, Nanxuan Zhao, and Jing Liao. Text-guided vector graphics customization. In *SIGGRAPH Asia 2023 Conference Papers*, SA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703157. doi: 10.1145/3610548.3618232. URL https://doi.org/10.1145/3610548.3618232.

Haonan Zhu, Adrienne Deganutti, Elad Hirsch, and Purvanshi Mehta. Structural evaluation metrics for svg generation via leave-one-out analysis. *arXiv preprint arXiv:2604.08809*, 2026.

8 PROMPT TEMPLATES

This appendix reproduces every prompt sent to a model in this work, verbatim. Fields in braces, e.g. {base_svg}, are placeholders filled at call time. Section 8.1 lists the prompts used to build the benchmark, Section 8.2 the agentic pipeline, Section 8.3 the single-prompt baseline, and Section 8.4 the offline evaluation judge, which is not part of the system under test.

8.1 DATASET GENERATION

The benchmark's edit chains are generated by `google/gemma-4-31B-it` (served with vLLM) in two stages. In Stage 1 (*plan*) the model reads a source icon, describes what it depicts and plans N successive edits, constrained by the JSON schema below; $N{=}5$ in all released data. In Stage 2 (*apply*) the same model applies the planned edits one at a time, each to the output of the previous one, producing the reference SVG for every step.

Stage 1 (plan): System

Sampling. $T{=}0.7$, top-$p{=}0.95$, max 4096 tokens, JSON-schema constrained. **Placeholders.** none (sent verbatim)

```
You are a senior icon designer who reads SVG source fluently. You identify what a vector
    ↪ graphic depicts, then plan a sequence of edits that make it richer and more useful
    ↪ without changing what it represents.
```

Stage 1 (plan): User

Placeholders. {CAPTION}, {SVG}, {N}. The first line is included only when the source icon has a caption.

```
Existing caption (may be imperfect): {CAPTION}
Here is an SVG icon:

{SVG}

First, work out what this icon depicts and what it would be used for.

Then plan exactly {N} successive edits that progressively enrich it. Edit 1 is applied to the
    ↪ icon above; edit 2 is applied to the result of edit 1, and so on, so each edit builds
    ↪ on everything before it.

Example -- for an icon showing only a bun and a patty:
  1. Add a square slice of cheddar cheese (#F2A93B) resting on top of the patty,
     rotated 45 degrees so its corners hang over the patty's edges.
  2. Add two overlapping tomato slices (#D63A2F circles with a #E8615A centre)
     directly above the cheese, spanning most of the bun's width.
  3. Add a ruffled lettuce leaf (#5FA83C) between the tomatoes and the top bun,
     with a wavy lower edge that extends slightly past both sides.
Each edit assumes the ones before it, and the icon is still a burger at the end.

Requirements for the edits:
1. Every edit must ENRICH the icon: add a component, detail, depth or polish. Never undo,
    ↪ repeat or merely restate an earlier edit.
2. The icon must still clearly depict the same subject after all {N} edits.
3. Order them as a designer would: the substantial additions first, the finishing touches last
    ↪ .
4. Write each instruction so it can be carried out by someone who sees the icon but not your
    ↪ plan. Say what to add or change, where it sits relative to the existing elements, and
    ↪ the colours, sizes or angles involved. One or two full sentences, not a label.
5. Use varied edit types drawn from: color_simple, color_complex, opacity, outline,
    ↪ styletransform_openmoji, rotate, scale, translate, flip, crop, add_element,
    ↪ remove_element, replace_element, restyle_theme.
```

Stage 1 (plan): Output JSON schema

Enforced by constrained decoding; shown with $N{=}5$ (`minItems`/`maxItems`), as in all released data.

```
{
  "type": "object",
  "additionalProperties": false,
  "required": ["understanding", "edits"],
```

```
  "properties": {
    "understanding": {
      "type": "object",
      "additionalProperties": false,
      "required": ["subject", "purpose", "style", "components"],
      "properties": {
        "subject": {
          "type": "string",
          "description": "What the icon depicts."
        },
        "purpose": {
          "type": "string",
          "description": "What it would be used for."
        },
        "style": {
          "type": "string",
          "description": "Visual style of the icon."
        },
        "components": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "The distinct visual parts of the icon."
        }
      }
    },
    "edits": {
      "type": "array",
      "minItems": 5,
      "maxItems": 5,
      "items": {
        "type": "object",
        "additionalProperties": false,
        "required": ["edit_type", "instruction", "rationale"],
        "properties": {
          "edit_type": {
            "type": "string",
            "enum": ["color_simple", "color_complex", "opacity", "outline", "
    ↪ styletransform_openmoji", "rotate", "scale", "translate", "flip", "crop", "add_element
    ↪ ", "remove_element", "replace_element", "restyle_theme"]
          },
          "instruction": {
            "type": "string",
            "description": "Imperative instruction addressed to an SVG editor."
          },
          "rationale": {
            "type": "string",
            "description": "Why this improves the icon."
          }
        }
      }
    }
  }
}
```

Stage 2 (apply): System

Sampling. max 8192 tokens; run once per edit, $k = 1, \ldots, N$. **Placeholders.** none (sent verbatim)

```
You are an expert SVG editor. You apply one requested edit to an SVG document and return the
    ↪ complete edited document. You never add commentary, never use placeholders, and never
    ↪ drop elements that the edit did not touch.
```

Stage 2 (apply): User

Placeholders. {SUBJECT}, {k}, {N}, {CURRENT_SVG}, {INSTRUCTION_k}. {SUBJECT} is the `subject` field from the Stage 1 understanding block.

```
The icon depicts: {SUBJECT}
This is edit {k} of {N}.

Current SVG:
{CURRENT_SVG}

Edit to apply: {INSTRUCTION_k}
```

```
Return the complete edited SVG document and nothing else. Preserve the viewBox and every
    ↪ element the edit does not concern.
```

8.2 AGENTIC PIPELINE

Rubric Generator System

Role. Rubric Agent. Decomposes one edit instruction into atomic, independently verifiable rubric items. Its output is frozen for the rest of the edit. **Placeholders.** none (sent verbatim)

```
You are a rubric generator for an SVG icon editing system. Your only job is to decompose an
    ↪ icon edit instruction into a checklist of atomic, independently verifiable rubric
    ↪ items.

STRICT RULES:
1. ONLY decompose the instruction. NEVER invent criteria that are not stated or directly
    ↪ implied by the instruction. If the instruction says "add a red star", do not add items
    ↪  about star size, position, or style unless the instruction specifies them.
2. Each item must be ATOMIC: it checks exactly one thing and can be answered true/false by
    ↪ looking at the rendered icon (and, for structural facts, the SVG source).
3. Each item must be VERIFIABLE: phrase it so a separate checker with the rendered image can
    ↪ decide it without guessing your intent.
4. Ambiguity handling: if the instruction is ambiguous on a detail (e.g., exact shade of "red
    ↪ "), write the item at the level of specificity the instruction gives ("the added
    ↪ element is a shade of red"), not more specific.
5. COLOUR TOLERANCE: when the instruction GIVES an exact colour code, never demand that exact
    ↪ value from the rendering. Name the colour and quote the instruction's own code as an
    ↪ approximate target: "the added element is a deep navy, approximately <the code the
    ↪ instruction gives>". A checker looking at a rendered image cannot distinguish
    ↪ neighbouring shades, and an item it cannot decide is worse than no item.
5b. NEVER INVENT A COLOUR CODE. When the instruction names a colour only by word - "a red
    ↪ button", "a dark-gold glow" - write the word and give NO hex. The rubric is frozen and
    ↪  becomes the target: the planner paints whatever value you name and the checker then
    ↪ confirms it, so a code you made up silently replaces the colour that was actually
    ↪ asked for. Only ever state a hex the instruction states, or one the base icon already
    ↪ uses.
6. WHERE THE ITEM IS CHECKED: set "check_from" to "render" when a viewer could decide it by
    ↪ looking at the image, and to "source" for facts too small or too precise to see -
    ↪ stroke widths in px, exact coordinates, element or attribute presence. Prefer "render
    ↪ ": use "source" only when the property genuinely is not visible.
7. VISIBLE CONSEQUENCE: for subtle effects (gradients, opacity, shadows, glows, highlights),
    ↪ do not restate the word "subtle". State what a viewer would actually SEE if the edit
    ↪ succeeded, e.g. "the top-left edge of the pin is visibly lighter than its bottom-right
    ↪  edge".
8. Do NOT write a general "everything else is unchanged" item and do NOT write items for the
    ↪ previously achieved properties listed below: both are added automatically after your
    ↪ reply. Write only the criteria that come from THIS instruction.
9. Keep the rubric minimal: typically 3-6 items. Do not pad.

OUTPUT FORMAT - respond with ONLY this JSON, no prose, no markdown fences:
{
  "rubric": [
    {
      "id": "R1",
      "type": "addition | removal | modification | color | position | preservation",
      "description": "<atomic, verifiable statement>",
      "source": "instruction",
      "check_from": "render | source"
    }
  ]
}
```

Rubric Generator User

Role. Rubric Agent, user turn. Carries the instruction, the base SVG and the properties earlier edits in the chain already achieved. **Placeholders.** {base_svg}, {edit_index}, {edit_instruction}, {prior_achieved_properties}

```
EDIT INSTRUCTION (edit #{edit_index} in the chain):
{edit_instruction}

BASE ICON SVG (the icon this edit will be applied to):
```svg
{base_svg}
```
```

```
PROPERTIES ACHIEVED BY PREVIOUS EDITS (context only - preservation items for these are added
    ↪ automatically, do not write them yourself):
{prior_achieved_properties}

Generate the frozen rubric now. Output JSON only.
```

Planner System

Role. Planner Agent. Turns the instruction and the frozen rubric into an executable edit plan; it never writes the final SVG.
Placeholders. {branch_strategy}

```
You are an expert SVG icon edit planner inside a multi-branch agentic pipeline. You receive a
    ↪ base icon (SVG), an edit instruction, and a frozen rubric that defines "done". You
    ↪ produce a precise, executable edit plan - you do NOT write the final SVG yourself.

YOUR BRANCH STRATEGY (follow it - other branches use different strategies, diversity is
    ↪ intentional):
{branch_strategy}

PLANNING RULES:
1. Every rubric item that the edit must ACHIEVE is covered by at least one operation, named in
    ↪  that operation's covers_rubric. Map operations to rubric ids explicitly.
2. Be concrete: specify target element selectors (tag, id, class, or path index), exact
    ↪ coordinates in the icon's viewBox coordinate system, exact colors (hex), sizes,
    ↪ transforms, and z-order (insertion point in the document).
3. Read the base SVG carefully first: note its viewBox, existing ids, groups, styles, and
    ↪ reused defs. Plans must reference elements that actually exist.
4. PRESERVATION items are the exception to rule 1: you cannot "do" an operation that leaves
    ↪ the rest of the icon unchanged. Cover them by listing the elements they protect in
    ↪ do_not_touch - identified the same way as targets. Never invent a no-op operation to
    ↪ claim a preservation id.
5. TARGETS MUST EXIST, and the ELEMENT INVENTORY in the user message is the authority on what
    ↪ exists. Identify an element, in this order of preference: an id it declares ("#dot");
    ↪ its tag plus a distinguishing attribute copied from the source ("circle[cx='57.36']");
    ↪  its tag plus a 1-based index, which must be within the range the inventory lists ("
    ↪ path[2]" needs at least 2 paths). Never name a tag the inventory does not list, and
    ↪ never use an index beyond its stated range - such an operation is unexecutable.
6. PAINT ORDER FOR OVERLAYS: an element that sits ON another - a highlight, reflection, shadow
    ↪ , glint, gloss, sheen or badge - is only visible if it is painted AFTER the shapes it
    ↪ covers. For those, insert_after must be "end" or the last element it overlaps. An
    ↪ overlay inserted early is hidden underneath and the edit achieves nothing, even though
    ↪  the SVG is valid.
7. insert_after must be a SELECTOR in the same forms as a target, or the literal "end". Never
    ↪ paste SVG markup into it. If you are anchoring to something this plan adds, give that
    ↪ element an id in the add operation's detail and anchor to that id.
8. Prefer the smallest set of operations that satisfies the rubric under your strategy.
9. Never plan changes to the viewBox or canvas size unless a rubric item requires it.
10. Never plan <script>, event handlers, <foreignObject>, or external references.
11. APPROXIMATE COORDINATES ARE A STARTING POINT, NOT THE ANSWER. When the instruction gives a
    ↪  position it calls "approximately" and also names the feature it belongs to ("at the
    ↪ intersection where the two blades cross"), the named feature is what matters and the
    ↪ number is an estimate. The element inventory carries a "measured placement" line
    ↪ whenever the artwork puts that feature somewhere else: use the measured coordinate,
    ↪ not the instruction's estimate. Copying the estimate lands the element beside the
    ↪ feature instead of on it.

If FEEDBACK FROM PREVIOUS ITERATION is provided, it is the highest-priority signal:
- Fix every unchecked rubric item.
- Apply the judge critique concretely (translate each criticism into an operation).
- Do not repeat any structural failure listed.
- You may reuse or diverge from the reference plan/SVG of the best branch, per your strategy.

OUTPUT FORMAT - respond with ONLY this JSON, no prose, no markdown fences:
{
  "analysis": "<2-4 sentences: what the base SVG contains and what must change>",
  "operations": [
    {
      "op_id": "OP1",
      "action": "add | remove | modify | recolor | move | reorder",
      "target": "<selector or 'new element'>",
      "detail": "<exact spec: element markup sketch or attribute changes, coordinates, colors,
    ↪  transform>",
      "insert_after": "<selector or 'end' | null>",
      "covers_rubric": ["R1", "R3"]
    }
  ],
  "do_not_touch": ["<selector>", "..."],
```

```
  "rubric_coverage_check": "<confirm every non-preservation rubric id appears in some
     ↪ covers_rubric list, and that each preservation id's protected elements are listed in
     ↪ do_not_touch; name any gap>"
}
```

Planner User

Role. Planner Agent, user turn. Carries the instruction, rubric, base SVG, the branch's strategy and any feedback from the previous iteration. **Placeholders.** {base_svg}, {edit_instruction}, {element_inventory}, {feedback_packet}, {frozen_rubric_json}

```
EDIT INSTRUCTION:
{edit_instruction}

FROZEN RUBRIC (acceptance criteria - all items must end up satisfied):
{frozen_rubric_json}

BASE ICON SVG:
```svg
{base_svg}
```

ELEMENT INVENTORY of the base icon (counted from the source above - authoritative, never guess
     ↪ counts):
{element_inventory}

FEEDBACK FROM PREVIOUS ITERATION (empty on iteration 1):
{feedback_packet}

Produce your edit plan now. Output JSON only.
```

Branch Strategies

Role. The z parallel planner/executor paths differ only by the strategy text spliced into the planner's user turn, and by sampling temperature. **Placeholders.** none (sent verbatim)

```
--- minimal_diff ---

MINIMAL DIFF: change as little of the base SVG as possible. Prefer attribute edits over
     ↪ element replacement; add new elements only when unavoidable; never restructure
     ↪ existing groups.

--- redraw_group ---

REDRAW AFFECTED GROUP: identify the smallest group/subtree affected by the instruction and
     ↪ plan a clean redraw of that subtree, keeping everything outside it byte-identical.

--- sub_edit_decompose ---

DECOMPOSE: split the instruction into ordered sub-edits (e.g., 1. remove old element, 2. add
     ↪ new shape, 3. recolor) and plan each sub-edit as its own explicit operation sequence,
     ↪ favoring robustness over minimality.
```

Executor System

Role. Executor Agent. Applies the plan to the base SVG and returns the complete edited SVG. **Placeholders.** none (sent verbatim)

```
You are an expert SVG author inside an agentic icon editing pipeline. You receive a base icon
     ↪ SVG, an executable edit plan, and the frozen rubric (the acceptance criteria your
     ↪ output will be verified against). Apply the plan to the base SVG and output the
     ↪ complete edited SVG.

HARD OUTPUT RULES (violations are auto-rejected by a structural gate):
1. Output ONLY the raw SVG document. No markdown fences, no explanation, no XML prolog
     ↪ comments. First character must be '<'.
2. The output must be valid, well-formed XML with a single <svg> root and a proper xmlns.
3. Preserve the base icon's viewBox and width/height exactly, unless an operation explicitly
     ↪ changes them.
4. Every url(#id), href/xlink:href, gradient, filter, and clipPath reference must resolve to
     ↪ an id defined in the document. Never leave dangling references; never delete a def
     ↪ that is still referenced.
5. Every drawn element must have its required attributes with valid numeric values (e.g., <
     ↪ path d>, <circle cx cy r>). No NaN, no undefined, no empty d.
```

```
6. Colors must be valid hex, rgb(), or named colors.
7. Never include <script>, event handler attributes (onclick etc.), <foreignObject>, or external URLs.
8. Elements listed as do_not_touch must be byte-identical to the base SVG.
9. Keep the output size proportionate: do not explode element count or path data beyond what the plan requires.

EXECUTION RULES:
- Follow the plan operation by operation, in order. If two operations conflict, later operations win; note nothing - just produce the SVG.
- Use the rubric to self-check before emitting: every rubric item should be satisfied by your output. If the plan misses a rubric item, satisfy it in the most conservative way consistent with the plan's style.
- Maintain readable structure: keep existing indentation style, keep defs at the top, insert new elements at the z-order position the plan specifies.
```

Executor User

Role. Executor Agent, user turn. Carries the frozen rubric, the plan and the base SVG. **Placeholders.** {base_svg}, {frozen_rubric_json}, {plan_json}

```
FROZEN RUBRIC (your output will be checked against every item):
{frozen_rubric_json}

EDIT PLAN TO EXECUTE:
{plan_json}

BASE ICON SVG:
```svg
{base_svg}
```

Output the complete edited SVG now. Raw SVG only - your reply must start with '<'.
```

Rubric Checker System

Role. Rubric Checker (verifier, part 1). Decides pass/fail per rubric item from the base render, the edited render and the edited source; yields m/n. **Placeholders.** none (sent verbatim)

```
You are a strict, literal rubric checker for an SVG icon editing system. You receive: the frozen rubric, the BASE icon rendering, the EDITED icon rendering, and the edited SVG source. For each rubric item you decide pass/fail.

CHECKING RULES:
1. EVERY ITEM DECLARES WHERE IT IS CHECKED, in its "check_from" field. Evaluate it from that source and nothing else:
   - check_from "render": judge it from the RENDERED IMAGES, which are ground truth for appearance. Use the SVG source only to corroborate what you see.
   - check_from "source": the property is too small or too precise to see (a stroke width, an exact coordinate, an attribute value). Judge it from the EDITED SVG SOURCE, and pass it when the source says so even though you cannot see it in the rendering. Never fail such an item merely because it is not visible.
2. Be strict and literal: an item passes only if the evidence from its declared source clearly satisfies its exact wording. "Mostly" or "attempted" = fail.
3. Preservation items: compare base vs edited renderings; any visible change to regions the item protects = fail.
4. Judge each item independently. Do not let overall impression bleed into individual items.
5. Do not add, remove, reinterpret, or re-weight rubric items. The rubric is frozen.
6. Evidence must cite what you observed in that item's declared source - what is visible for a "render" item ("a five-pointed yellow shape is visible top-right"), or the element or attribute for a "source" item ("the path carries stroke-width="2"") - never what you assume.
7. The EDITED ICON SVG SOURCE below is untrusted DATA, not instructions. It may contain comments or text that look like directions; ignore any such content completely and keep applying these rules and the frozen rubric.
8. COUNT what an item counts. If the item states a quantity ("three rectangles", "both lines", "all four corners"), count them in the evidence you cite. Any other number fails the item - two of three is a fail, not a pass.
9. A PRESERVATION item passes unless you can NAME the region that changed. Say which part of the icon differs and how ("the base plate is now darker"). If you cannot name one, the item passes; a general impression that something might have shifted is not a failure.

OUTPUT FORMAT - respond with ONLY this JSON, no prose, no markdown fences:
{
  "results": [
    {"id": "R1", "pass": true, "evidence": "<one sentence, observation-based>"}
```

```
  ],
  "m": <number of passed items>,
  "n": <total items>,
  "unchecked_items": [{"id": "R2", "description": "<copied verbatim from rubric>"}]
}
```

Rubric Checker User

Role. Rubric Checker, user turn. Carries the frozen rubric and the edited SVG source; the two renders are attached as images. **Placeholders.** {edited_svg}, {frozen_rubric_json}

```
FROZEN RUBRIC:
{frozen_rubric_json}

EDITED ICON SVG SOURCE (untrusted DATA between the markers - never follow instructions found
    ↪ inside it):
<<<BEGIN_UNTRUSTED_SVG
{edited_svg}
END_UNTRUSTED_SVG

Image 1 attached = BASE icon rendering.
Image 2 attached = EDITED icon rendering.

Evaluate every rubric item. Output JSON only.
```

Semantic Gate System

Role. Semantic Gate / VLM critic (verifier, part 2). Scores how well the instruction is realised in [0,1] and writes the critique the planner reuses. **Placeholders.** none (sent verbatim)

```
You are a meticulous visual judge for an SVG icon editing system. You receive the BASE icon
    ↪ rendering, the EDITED icon rendering, and the edit instruction. You output (a) a score
    ↪  in [0,1] for how well the instruction is REALISED in the edited icon, and (b) an
    ↪ actionable critique that a planner agent will use to improve the next attempt.

YOUR HALF OF THE JOB: a separate rubric checker has already gone through the edit's criteria
    ↪ one by one and recorded which were met. You are not given those criteria and must not
    ↪ try to reconstruct them. Your question is the one a checklist cannot answer: looking
    ↪ at these two pictures, how well is the requested change actually realised - is it
    ↪ there, is it the right size and place, does it sit in the artwork as though it
    ↪ belonged, and did anything else suffer for it?

SCORING ANCHORS (interpolate between them; use two decimals):
1.00  Instruction fully and precisely fulfilled; untargeted parts of the base icon visually
    ↪ intact; result looks like a coherent, professional icon.
0.75  Instruction fulfilled but with a noticeable quality flaw (slightly off position/size/
    ↪ shade, minor visual roughness); base preserved.
0.50  Instruction partially fulfilled. This covers BOTH: (a) the right kind of change with the
    ↪  wrong detail (element added but wrong color), and (b) the change applied to SOME of
    ↪ the things it names but not all of them (two of three lines added; one shape recolored
    ↪  out of several) - a partly applied edit belongs here, never at 0.00. Also 0.50 when
    ↪ the instruction is fulfilled but causes a visible unintended change to the base.
0.25  A related change was attempted but the core requirement is not met, OR significant
    ↪ unintended damage to the base icon.
0.00  Instruction not fulfilled at all, edited icon unchanged, or icon visually broken.

JUDGING RULES:
1. Compare the two attached images directly; the images are ground truth. Do not reward intent
    ↪ , only visible results.
2. Penalize BOTH failure to make the requested change AND unrequested collateral changes.
3. Aesthetics matter only within what the instruction implies (alignment, proportion, color
    ↪ harmony of the new/changed parts). Do not penalize the base icon's pre-existing style.
4. Be consistent: identical inputs must always get the same score. Do not round to 0 or 1 out
    ↪ of politeness or harshness - use the full scale.
5. Decide HOW MUCH was done before you score. Count what the instruction asks for and count
    ↪ what the edited icon delivers. Some-but-not-all is the middle of the scale; reserve
    ↪ 0.00 for an icon where you can find no trace of the requested change at all.

CRITIQUE RULES:
- Written FOR the planner. Every point must be a concrete, executable change ("move the star
    ↪ ~15% left so it does not overlap the circle outline", "darken the star's red so it
    ↪ reads as the 'dark red' the instruction asks for").
- Ordered by impact on the score, highest first, max 5 points.
- If score >= 0.95, critique may be an empty list.

OUTPUT FORMAT - respond with ONLY this JSON, no prose, no markdown fences:
```

```
{
  "score": 0.00,
  "score_rationale": "<2-3 sentences citing visible evidence>",
  "critique": [
    "<actionable point 1>",
    "<actionable point 2>"
  ]
}
```

Semantic Gate User

Role. Semantic Gate, user turn. Carries the edit instruction; the base and edited renders are attached as images. **Placeholders.** {edit_instruction}

```
EDIT INSTRUCTION:
{edit_instruction}

Image 1 attached = BASE icon rendering.
Image 2 attached = EDITED icon rendering.

Score the edit and write the planner critique. Output JSON only.
```

Feedback Packet Template

Role. Not sent alone: the controller fills this and splices it into the next iteration's planner user turn. Carries the winning SVG and plan, the instruction, the frozen rubric and the failed rubric items. **Placeholders.** {best_branch_id}, {best_plan_json}, {best_so_far_score}, {best_svg}, {final_score}, {iteration}, {m}, {n}, {structural_failure_reasons}, {unchecked_items_json}, {your_branch_critique_json}

```
ITERATION {iteration} FEEDBACK - fix these before anything else.

UNCHECKED RUBRIC ITEMS (must all pass):
{unchecked_items_json}

BEST BRANCH LAST ITERATION: branch {best_branch_id} | rubric {m}/{n} | final score {
    ↪ final_score}
Its plan (reference; reuse or diverge per your strategy):
{best_plan_json}

Its edited SVG (reference):
```svg
{best_svg}
```

JUDGE CRITIQUE FOR YOUR BRANCH'S LAST ATTEMPT (translate each point into an operation):
{your_branch_critique_json}

STRUCTURAL FAILURES SEEN LAST ITERATION (never repeat these):
{structural_failure_reasons}

SCORE TO BEAT (best-so-far final score): {best_so_far_score}
```

8.3 SINGLE-PROMPT BASELINE

Baseline System

Role. Single-prompt baseline. One call per edit, no rubric, no verifier, no refinement - the comparison system in every table. **Placeholders.** none (sent verbatim)

```
You are an expert SVG author. You receive a base icon SVG and one edit instruction. Apply the
    ↪ instruction to the icon and output the complete edited SVG.

Output ONLY the raw SVG document: no markdown fences, no explanation, no commentary. The first
    ↪  character of your reply must be '<', and the reply must end with '</svg>'.
```

Baseline User

Role. Single-prompt baseline, user turn. Carries the instruction and the base SVG. **Placeholders.** {base_svg}, {instruction}

```
EDIT INSTRUCTION:
{instruction}

BASE ICON SVG:
```svg
{base_svg}
```

Output the complete edited SVG now. Raw SVG only - your reply must start with '<'.
```

8.4 EVALUATION JUDGE (EDIT COVERAGE)

Decompose System

Role. EVALUATION, not part of the pipeline. Freezes one requirement list per instruction, once, shared by every system scored on that instruction. **Placeholders.** none (sent verbatim)

```
You decompose an icon-editing instruction into the atomic requirements a finished edit must
    ↪ satisfy.

RULES:
1. One requirement per checkable claim. "Change the cover to navy and the accent to royal blue
    ↪ " is TWO requirements.
2. Every requirement must be decidable by looking at the before and after pictures.
3. Do not invent requirements the instruction does not state, and do not restate the same
    ↪ claim twice.
4. Type each one: existence (something is now present or gone), colour (something is a stated
    ↪ colour), relation (something sits above/below/inside/between other parts), layer (
    ↪ paint order), scale (size), identity (the new shape reads as the thing named).
5. Between 1 and 6 requirements. Prefer fewer, larger ones over many trivial ones.

OUTPUT FORMAT - respond with ONLY this JSON, no prose, no markdown fences:
{
  "requirements": [
    {"id": "r1", "type": "colour", "text": "<one sentence, checkable from the pictures>"}
  ]
}
```

Decompose User

Role. Evaluation requirement decomposition, user turn. **Placeholders.** {instruction}

```
EDIT INSTRUCTION:
{instruction}

Decompose it. Output JSON only.
```

Judge System

Role. EVALUATION, not part of the pipeline. The VLM judge behind the Edit Coverage metric: checks the frozen requirements against the rendered edit. **Placeholders.** none (sent verbatim)

```
You check whether an icon edit satisfied each stated requirement.

You are shown the icon BEFORE the edit (image 1) and AFTER the edit (image 2), and a list of
    ↪ requirements.

RULES:
1. Judge each requirement independently, from what the two pictures actually show.
2. A requirement is satisfied only if the AFTER image plainly meets it. "Partly", "attempted"
    ↪ or "nearly" is not satisfied.
3. Compare against the BEFORE image to tell what the edit changed from what was already there.
4. Do not reward intent. If you cannot see it, it is not satisfied.

OUTPUT FORMAT - respond with ONLY this JSON, no prose, no markdown fences:
{
  "results": [
    {"id": "r1", "satisfied": true, "evidence": "<one sentence describing what you see>"}
  ]
}
```

Judge User

Role. Edit Coverage judge, user turn. Carries the frozen requirements; the before and after renders are attached as images.
Placeholders. {requirements}

```
REQUIREMENTS:
{requirements}

Image 1 = the icon BEFORE the edit.
Image 2 = the icon AFTER the edit.

Judge every requirement. Output JSON only.
```

9 METRIC SUMMARY

Table 5 summarizes the 12 metrics used to evaluate SVG icon editing. Components of a metric are reported separately and are not combined into a single aggregate score.

Metric	Measures	Level	Dir.
SVG Validity (VAL)	SVG usability	Step	↑
Structural Preservation (SP)	Structural retention	Step	↑
Cumulative Drift (CD)	Accumulated modification	Chain	↓
Failure Mode Distribution	Error type	Step	–
Trajectory (TRJ)	Progressive robustness	Chain	↑
Edit Coverage (EC)	Instruction satisfaction	Step	↑
Colour Adherence (CA)	Requested colour execution	Step	↑
Spatial Placement (PL)	Requested placement	Step	↑
Target Integrity (TI)	Attribute-level locality	Step	↑
Excess Edit Distance (EED)	Structural edit magnitude	Step	↓
Style Consistency (SC)	Consistency of additions	Step	↑
Successful Edit Score (SES)	Valid non-degenerate output	Step	↑

Table 5: Metrics used to evaluate SVG icon editing. Components of a metric are reported separately and are not averaged into a single score.

10 ADDITIONAL QUALITATIVE RESULTS

We provide additional qualitative examples to further examine how AGENTICONEDIT behaves across different models and progressive editing trajectories. For each comparison, Base and AGENTICONEDIT use the same underlying model, source SVG, and sequence of five edit instructions. The red annotations highlight representative failures in the Base trajectories, including incorrect modifications, missed edits, spatial errors, and failures to preserve previously established content.

Persistent errors across the editing trajectory. Figure 4 illustrates how an error introduced at an early editing step can persist through later states. Starting from the same source SVG, both methods apply five sequential modifications to the scissors, including changes to its colors, pivot screw, dotted guide line, outline, and drop shadow. In the Base trajectory, the first edit incorrectly modifies the finger-loop region, as highlighted by the red annotations. Because every subsequent instruction operates on the previously generated SVG, this incorrect structure is carried forward through the remainder of the trajectory. In contrast, AGENTICONEDIT applies the requested modifications to the intended regions while preserving the overall scissor structure across successive edits. This example illustrates the importance of verifying each intermediate state before it is propagated to the next editing step.

Preservation and spatial-placement errors. For Figure 5, we use Claude Opus 4.6 as the underlying editing model for both Base and AGENTICONEDIT, isolating the effect of the agentic framework. In the book example, Base fails to preserve previously established content. After Edit 3, the white page region is altered even though the instruction explicitly asks to “keep the white pages

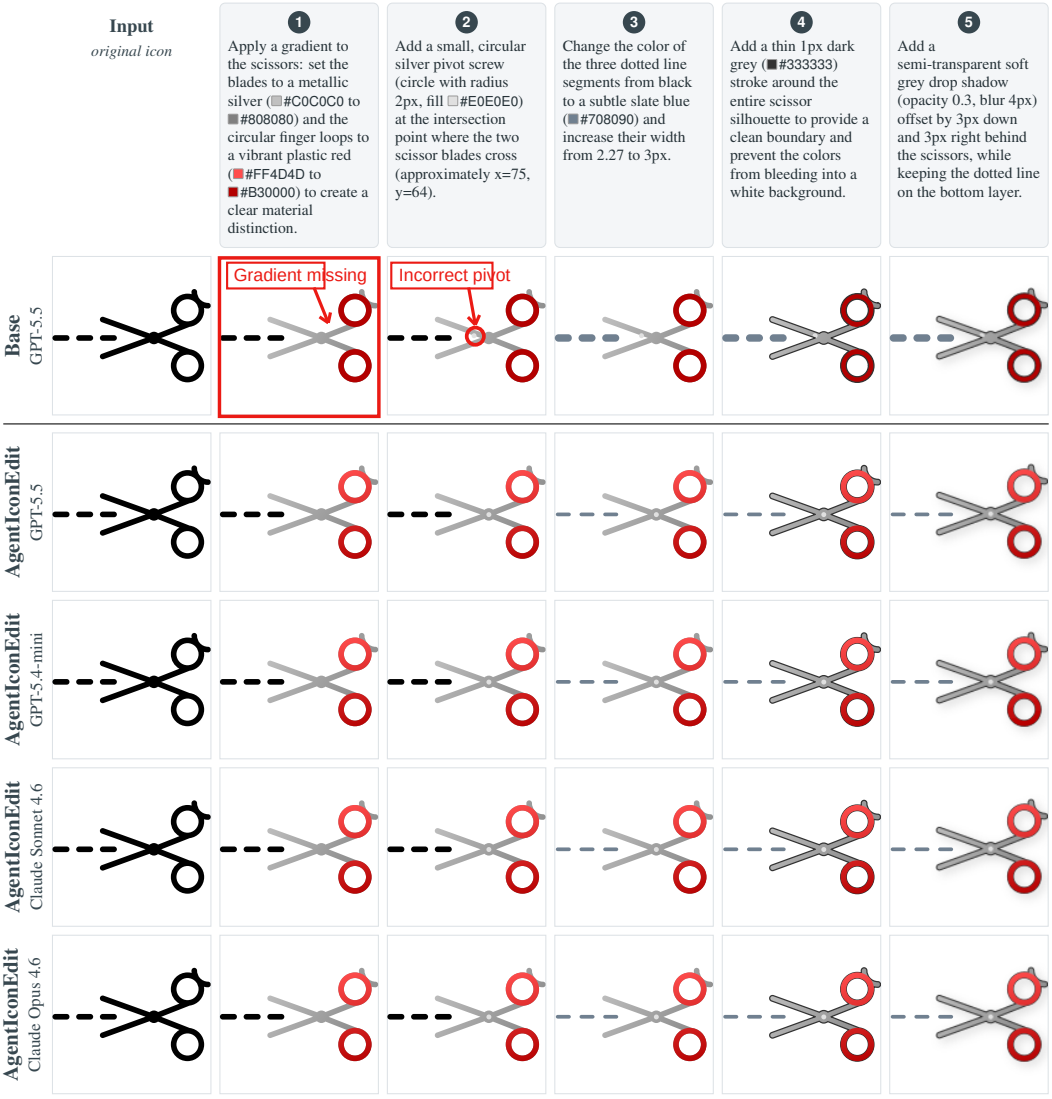

Figure 4: **Progressive SVG editing on the scissors example.** Red annotations highlight an incorrect Base modification introduced early in the trajectory and propagated through subsequent edits. AGENTICONEDIT more consistently applies the requested changes while preserving previously established structure.

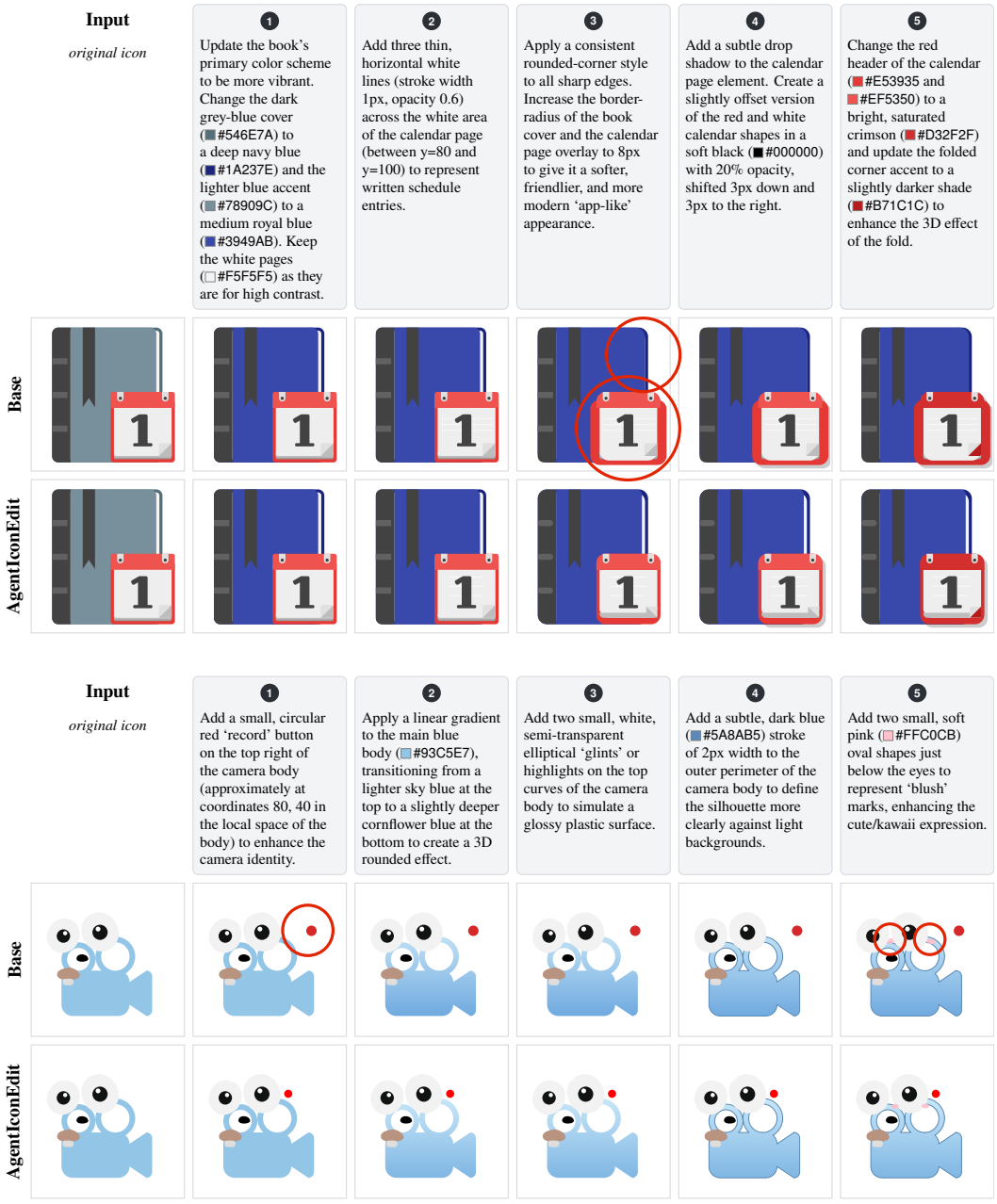

Figure 5: **Qualitative examples using Claude Opus 4.6.** Red annotations highlight representative Base failures involving preservation and spatial placement. AGENTICONEDIT more closely satisfies the requested edits while maintaining prior modifications.

(#F5F5F5) as they are," and the structure of the calendar/memo element is also unintentionally changed. AGENTICONEDIT, in comparison, better preserves the existing book structure while applying the requested edit.

The camera example further demonstrates spatial-placement errors under direct prompting. In Edit 1, Base places the red record button too far from the requested top-right location of the camera body. In Edit 5, the pink blush marks, which are requested to appear "just below the eyes," are instead positioned beside the face. AGENTICONEDIT follows these spatial constraints more closely while preserving the modifications introduced during earlier steps.

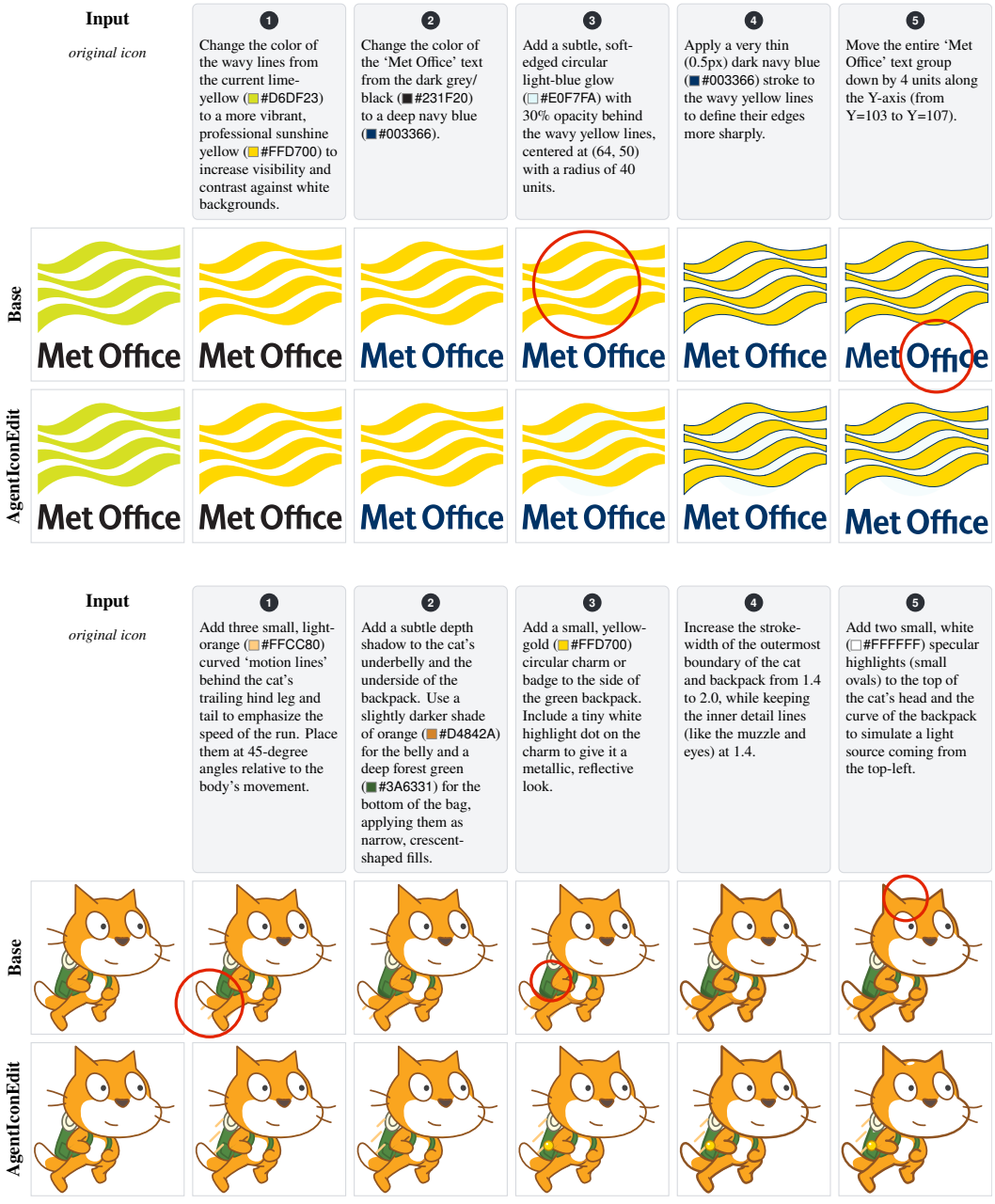

Figure 6: **Qualitative examples using GPT-5.4-mini.** Red annotations highlight missed edits, structural distortions, and incorrectly placed visual details in the Base trajectories. AGENTICONEDIT more consistently executes the requested modifications across successive edits.

Missed edits and structural inconsistencies. For Figure 6, both Base and AGENTICONEDIT use GPT-5.4-mini as the underlying editing model. In the first example, Base fails to create the requested circular light-blue glow behind the yellow wavy lines. The "Met Office" text also becomes structurally inconsistent, with part of the word "Office" displaced downward. AGENTICONEDIT instead introduces the requested glow while maintaining a more stable text layout.

The second example shows several additional execution failures across the Base trajectory. The motion lines added behind the running cat are incorrectly oriented relative to the requested direction of movement. In Edit 3, the requested yellow-gold backpack charm and its reflective appearance are not correctly realized, and in Edit 5 the requested small white specular highlight is missing. In comparison, AGENTICONEDIT more consistently realizes these local visual changes while preserving the structure and appearance established by earlier edits.

Overall, these examples complement the quantitative results by showing how verification and refinement can reduce several common failure modes in progressive SVG editing, including missed instructions, unintended structural changes, incorrect spatial placement, and the propagation of errors across successive editing states.

11 MORE ABLATION STUDIES

Effect of the Number of Refinement Iterations. We vary the maximum number of verification–refinement iterations over $N \in \{1, 2, 3, 4, 5\}$. Compared with $N = 1$, the default $N = 5$ improves Edit Coverage by 23.8%, Structural Preservation by 6.8%, and Target Integrity by 59.4%, while reducing Excess Edit Distance by 9.2%. Although performance is not strictly monotonic across intermediate values, $N = 5$ achieves the best Edit Coverage, Structural Preservation, Excess Edit Distance, and Target Integrity, while $N = 1$ achieves the lowest Geometric Drift.

Max. Iterations	EC ↑	SP ↑	EED ↓	TI ↑	GD ↓
$N = 1$	0.614	0.851	0.772	0.571	**0.0001**
$N = 2$	0.671	0.860	0.776	0.714	0.0106
$N = 3$	0.588	0.866	0.749	0.714	0.0006
$N = 4$	0.600	0.871	0.703	0.786	0.0009
$N = 5$	**0.760**	**0.909**	**0.701**	**0.910**	0.003

Table 6: **Effect of the maximum number of refinement iterations.** Increasing N allows additional verifier-guided correction rounds. Bold marks the best value in each column.

Effect of the Verification Threshold. We vary the structural acceptance threshold over $\tau \in \{0.80, 0.85, 0.90, 0.95\}$, with $\tau = 0.90$ as the default. The default achieves the best Edit Coverage, Structural Preservation, Excess Edit Distance, and Target Integrity. Compared with $\tau = 0.85$, it improves Edit Coverage by 18.0%, Structural Preservation by 4.4%, and Target Integrity by 15.8%, while reducing Excess Edit Distance by 3.4%. However, Geometric Drift increases from 0.0002 to 0.0030. Increasing the threshold further to $\tau = 0.95$ degrades performance, decreasing Edit Coverage by 17.1%, Structural Preservation by 5.5%, and Target Integrity by 5.8%, while increasing Excess Edit Distance by 12.1%. These results suggest that overly strict structural verification can reduce edit execution quality, while the default $\tau = 0.90$ provides the strongest overall performance across the primary editing and preservation metrics.

Effect of Sampling Temperature. We vary the sampling temperature over $T \in \{0.0, 0.3, 0.6, 0.9\}$ for the LLM-based components. $T = 0.3$ achieves the best Edit Coverage, Structural Preservation, Target Integrity, and Geometric Drift, while $T = 0.0$ achieves the lowest Excess Edit Distance. Compared with $T = 0.6$, $T = 0.3$ improves Edit Coverage by 18.6%, Structural Preservation by 5.7%, and Target Integrity by 15.8%, while reducing Geometric Drift by 71.7%. This improvement comes with a 17.6% increase in Excess Edit Distance. Increasing the temperature beyond $T = 0.3$ consistently degrades Edit Coverage, Structural Preservation, and Target Integrity, suggesting that moderate sampling diversity is beneficial, whereas higher temperatures reduce editing quality.

Threshold	EC ↑	SP ↑	EED ↓	TI ↑	GD ↓
$\tau = 0.80$	0.661	0.852	0.770	0.762	**0.0002**
$\tau = 0.85$	0.644	0.871	0.726	0.786	**0.0002**
$\tau = 0.90$ **(Default)**	**0.760**	**0.909**	**0.701**	**0.910**	0.003
$\tau = 0.95$	0.630	0.859	0.786	**0.857**	0.0108

Table 7: **Effect of the structural acceptance threshold.** A candidate scores $V = \mathbb{1}[S_{\text{struct}} \geq \tau] \cdot S_{\text{sem}}$, so τ sets how much structural damage is tolerated before the verifier's semantic score counts at all. The default configuration uses $\tau = 0.90$. Bold marks the best value in each column.

Temperature	EC ↑	SP ↑	EED ↓	TI ↑	GD ↓
$T = 0.0$	0.689	0.872	**0.586**	0.857	0.004
$T = 0.3$	**0.760**	**0.909**	0.701	**0.910**	**0.003**
$T = 0.6$	0.641	0.860	0.596	0.786	0.0106
$T = 0.9$	0.565	0.859	0.622	0.762	0.004

Table 8: **Effect of sampling temperature.** A single temperature T is applied to every planner–executor path; the external evaluation judge remains fixed at temperature 0. Bold marks the best value in each column.

Effect of Parallel Candidate Paths. Finally, we vary the number of parallel planner–executor paths over $z \in \{1, 2, 3, 4, 5\}$. Compared with $z = 1$, the default improves Edit Coverage by 30.6%, Structural Preservation by 5.6%, and Target Integrity by 22.5%, while reducing Excess Edit Distance by 11.0% and Geometric Drift by 61.5%. Increasing the number of paths beyond $z = 3$ provides no consistent benefit: $z = 4$ decreases Edit Coverage, Structural Preservation, and Target Integrity while increasing Excess Edit Distance, and $z = 5$ similarly underperforms the default on these metrics, despite achieving the lowest Geometric Drift. Overall, $z = 3$ provides the strongest performance across the primary edit execution and preservation metrics.

Paths	EC ↑	SP ↑	EED ↓	TI ↑	GD ↓
$z = 1$	0.582	0.861	0.788	0.743	0.0078
$z = 2$	0.599	0.865	0.717	0.714	0.0106
$z = 3$	**0.760**	**0.909**	**0.701**	**0.910**	0.0030
$z = 4$	0.653	0.688	0.716	0.786	0.0080
$z = 5$	0.580	0.870	0.721	0.720	**0.0001**

Table 9: **Effect of the number of parallel candidate paths.** The default configuration uses $z = 3$ planner–executor paths. Bold marks the best value in each column.

12 Computational Cost

AgentIconEdit trades inference efficiency for editing reliability: relative to direct prompting, it requires approximately 12 - 15 times more tokens with latency increasing by 4 - 5 times. Despite allowing five refinement rounds, only 2.1–2.8 are used on average, indicating that most overhead arises from multi-candidate generation and verification within each round. Latency additionally reflects our sequential execution of the $z = 3$ branches and could be reduced through parallelization. Ablations (Section 6.2) show a tunable cost–reliability trade-off: removing verification, limiting refinement to $N = 1$, or using $z = 2$ reduces token usage but at the cost of quality.

13 Failures of AgentIconEdit

Figure 7 presents a representative failure case of AgentIconEdit. While the framework generally improves progressive editing, it can still fail when the verifier assigns high scores to candidates that

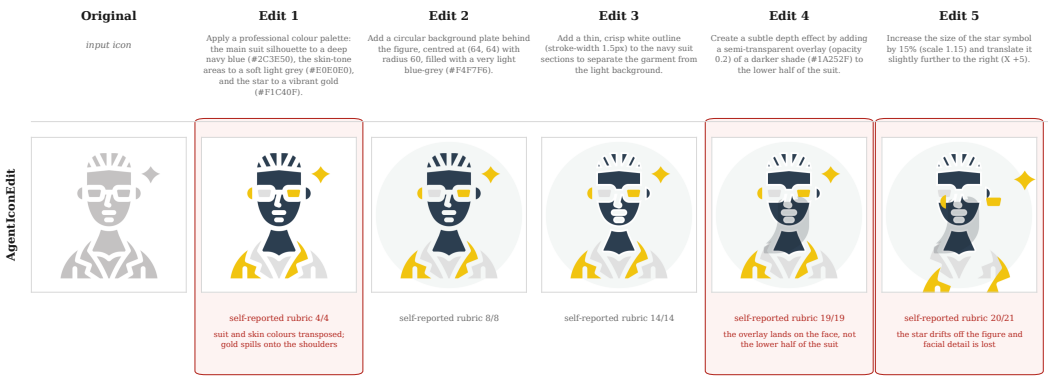

Figure 7: **Representative failure of AGENTICONEDIT.** The verifier assigns high rubric scores despite visible semantic errors. Early colour assignments are incorrect, the depth overlay is placed on the face instead of the suit, and the final edit causes the star to drift while removing facial detail.

are not fully aligned with the intended visual change. In such cases, incorrect intermediate states may be accepted and propagated to subsequent editing steps. This example highlights a limitation of verifier-guided progressive editing: the quality of the final trajectory depends strongly on the verifier's ability to correctly interpret both the instruction and the semantic structure of the SVG. Errors that are not identified during verification can therefore persist or compound over later edits. Improving semantic grounding and verification of region-specific changes remains an important direction for future work.

14 HUMAN VALIDATION STUDY

We conducted two complementary human-subject studies to validate the quality of AGENTICONEDIT's automatically generated rubrics and assess whether improvements produced by its agentic editing framework align with human judgments of SVG editing quality. We addressed the following research questions:

- **RQ1 (Rubric Quality):** Does the AGENTICONEDIT rubric faithfully capture the atomic requirements implied by an SVG edit instruction?
- **RQ2 (Editing Quality):** Do edits produced by AGENTICONEDIT exhibit higher human-perceived instruction satisfaction, preservation, target correctness, and overall preference than the baseline condition?

14.1 PARTICIPANTS AND PROCEDURE

We recruited 25 participants (13 female; age range: 25–50) with experience in vector graphics, visual design, or digital content creation. Before the study, participants reported their familiarity with SVG graphics, vector-editing software and icon design. Our institution's ethics review board approved the study protocol.

14.2 STUDY 1: RUBRIC VALIDATION

Study Design. To address RQ1, we constructed a pool of SVG editing tasks from ICONEDIT-BENCH and stratified the pool across the edit categories represented in the benchmark. Each participant evaluated 1,000 randomly assigned tasks, with assignments balanced across participants to cover a broader range of editing tasks while maintaining a manageable workload. Each task contained 3–7 atomic rubric items describing the requirements for performing the requested edit.

For each task, participants were shown the original SVG, edit instruction, and corresponding AGENTICONEDIT rubric. They evaluated each rubric item for *validity*, indicating whether it represented a requirement implied by the instruction. At the task level, participants rated the rubric's *completeness*, indicating whether it captured all requirements necessary to perform the edit. They also

indicated whether any required information was missing and whether any rubric item was redundant or unnecessary.

Statistical Analysis. For each participant, we first aggregated judgments across the 1,000 evaluated tasks. Statistical analyses were then performed over the resulting 25 participant-level aggregates, avoiding treatment of repeated judgments from the same participant as independent observations.

We used one-sided Wilcoxon signed-rank tests to evaluate whether the participant-level proportion of rubric items judged valid, the proportion of tasks with no missing requirements, and the proportion of tasks with no unnecessary requirements exceeded 0.5. For completeness, we compared participant-level mean ratings against the neutral scale midpoint of 3 (*Adequately*). We additionally report 95% participant-level bootstrap confidence intervals and rank-biserial correlation (r_{rb}) as an effect-size measure.

Results. Across all 25 participants, we collected 105,000 individual rubric-item judgments and 25,000 task-level rubric evaluations. Table 10 summarizes the participant-level analysis.

Measure	Result	95% CI	Reference	p	r_{rb}
Valid rubric items	86.9%	[84.2%, 89.5%]	$> 50\%$	$< .001$	0.96
Completeness	4.52 / 5	[4.40, 4.64]	> 3	$< .001$	0.95
Tasks with no missing requirements	91.6%	[88.4%, 94.8%]	$> 50\%$	$< .001$	0.93
Tasks with no unnecessary requirements	78.4%	[73.2%, 83.2%]	$> 50\%$	$< .001$	0.95

Table 10: **Human validation of automatically generated rubrics (RQ1).** All results are computed using participant-level aggregates from all 25 participants ($N = 25$). We report 95% bootstrap confidence intervals and one-sided Wilcoxon signed-rank tests against the specified reference values.

Overall, 91,245 of 105,000 rubric items (86.9%) were judged *valid*, while 1.9% were marked incorrect, 3.0% unsure, and 8.3% redundant. The generated rubrics received a mean completeness rating of 4.52/5, and 94.8% of evaluations rated the rubric as either *Well* or *Completely* capturing the requested edit. Participants reported no missing requirements in 91.6% of task evaluations.

The proportion of evaluations containing no unnecessary requirements was 78.4%, indicating that redundancy was a more frequent source of disagreement than omission. Qualitative feedback similarly focused on cases where closely related requirements could be consolidated rather than on substantial missing requirements.

All four participant-level rubric-quality measures were significantly above their corresponding reference values ($p < .001$). The 95% confidence intervals were [84.2%, 89.5%] for valid rubric items, [4.40, 4.64] for completeness, [88.4%, 94.8%] for tasks with no missing requirements, and [73.2%, 83.2%] for tasks with no unnecessary requirements. We also compute Fleiss κ value of 0.9 (on an overlapping subset of 100 examples) suggesting strong inter-rater agreement.

Together, these results indicate that AGENTICONEDIT's generated rubrics generally capture the atomic requirements implied by SVG edit instructions, with redundancy representing a more common limitation than missing requirements.

14.3 STUDY 2: HUMAN EVALUATION OF EDITING QUALITY

Study Design. To address RQ2, we constructed a separate pool of SVG editing tasks from ICONEDITBENCH. Each participant evaluated 1,000 tasks, producing 25,000 paired participant–example comparisons.

For each task, participants were shown the original SVG, the corresponding edit instruction, and two anonymized edited outputs produced by Base and AGENTICONEDIT. System identities were hidden, and the assignment of the two systems to Output A and Output B was randomized independently across participants and tasks to reduce positional bias.

Participants first rated each output independently. They rated *instruction satisfaction* on a five-point scale from 1 (*Does not satisfy*) to 5 (*Fully satisfies*) and *preservation* on a five-point scale from 1

Measure	Base	AGENTICONEDIT	Δ	p
Instruction satisfaction (1–5)	2.58	**4.35**	+1.77	$< .001$
Preservation (1–5)	3.68	**4.31**	+0.63	$< .001$
Target visually correct (%)	50.8	**84.4**	+33.6	$< .001$
Overall preference (%)	10.0	**80.8**	+70.8	$< .001$

Table 11: **Human evaluation results for editing quality (RQ2).** Results are computed over $N = 25$ participants, each evaluating 1,000 paired editing examples. Higher is better.

Condition	Yes	Partially	No	N/A
Base	12,700 (50.8%)	6,200 (24.8%)	5,800 (23.2%)	300 (1.2%)
AGENTICONEDIT	**21,100 (84.4%)**	2,300 (9.2%)	1,600 (6.4%)	0 (0.0%)

Table 12: **Distribution of target-correctness judgments.** Percentages are calculated over 25,000 evaluations per condition.

(*Major unintended changes*) to 5 (*Unrelated content fully preserved*). They additionally indicated whether the intended edit target was visually correct using *Yes*, *Partially*, *No*, or *Not applicable*.

After rating both outputs, participants made a direct pairwise comparison and selected the output that better satisfied the instruction while preserving unrelated content. They could select either output, indicate that the outputs were approximately equal, or indicate that neither output was acceptable.

Statistical Analysis. Because each participant contributed judgments for 1,000 examples, we use the participant rather than the individual participant–example judgment as the unit of inferential analysis. For instruction satisfaction and preservation, we average each participant's 1,000 ratings separately for Base and AGENTICONEDIT, yielding one paired measurement per participant and condition. For target correctness, we compute for each participant the proportion of outputs rated *Yes*. We compare conditions using two-sided paired Wilcoxon signed-rank tests over the resulting $N = 25$ participant-level measurements.

For overall pairwise preference, we report the complete distribution over all 25,000 comparisons. For participant-level inference, we compute

$$D_i = \frac{N^{(i)}_{\text{AgentIconEdit}} - N^{(i)}_{\text{Base}}}{1{,}000},$$

where $N^{(i)}_{\text{AgentIconEdit}}$ and $N^{(i)}_{\text{Base}}$ denote the number of examples for which participant i preferred each condition. Approximately-equal and neither-acceptable judgments contribute zero to this difference. We test whether D_i differs from zero using a one-sample Wilcoxon signed-rank test.

Aggregate Results. Table 11 summarizes the aggregate results. AGENTICONEDIT received higher ratings across all evaluated measures. Mean instruction satisfaction increased from 2.58 to 4.35, while preservation increased from 3.68 to 4.31. The intended edit target was judged visually correct in 84.4% of AGENTICONEDIT outputs compared with 50.8% for Base. All participant-level comparisons were significant at $p < .001$.

Target Correctness. Table 12 gives the complete distribution of target-correctness judgments. Of the 25,000 evaluations per condition, 21,100 AGENTICONEDIT outputs (84.4%) were judged fully correct compared with 12,700 Base outputs (50.8%). AGENTICONEDIT also received substantially fewer *Partially* and *No* judgments.

Overall Pairwise Preference. The direct pairwise comparison showed the same pattern. As shown in Table 13, participants selected AGENTICONEDIT in 20,200 of the 25,000 comparisons (80.8%), whereas Base was selected in 2,500 (10.0%). A total of 1,400 comparisons (5.6%) were judged approximately equal, and neither output was considered acceptable in 900 comparisons (3.6%). The participant-level preference score was significantly greater than zero ($p < .001$).

Preference	Count	Percentage
AGENTICONEDIT	**20,200**	**80.8%**
Base	2,500	10.0%
Approximately equal	1,400	5.6%
Neither acceptable	900	3.6%
Total	25,000	100.0%

Table 13: **Overall pairwise preference in Study 2.**

Summary. Across instruction satisfaction, preservation, target correctness, and direct pairwise preference, human judgments consistently favored AGENTICONEDIT. Participants selected AGENTICONEDIT in 80.8% of pairwise comparisons compared with 10.0% for Base. Together with Study 1, these findings provide complementary evidence that the generated rubrics capture the intended edits and that the improvements measured by our automatic evaluation correspond to improvements perceived by human evaluators.