SCALPEL: Specification-Constrained Agents Learning Precise Edit Localization

Anonymous EMNLP submission

Abstract

When a language model edits a structured file, such as an SVG image, a piece of code, or a UI layout, it often satisfies the requested change while quietly breaking something it was never asked to touch. We call this **quiet corruption**. The problem is invisible to standard benchmarks, which score only whether the model did the right thing and not whether it left everything else alone.

We introduce **SVGym**, a benchmark of 106 SVG editing tasks spanning 19 fault categories and 4 difficulty tiers. Unlike existing benchmarks, SVGym measures two independent quantities: *instruction adherence* (did the model make the requested edit?) and *preservation* (did it leave the rest of the file intact?). Frontier models top out at 29.2% exact solve, a gap driven by the mismatch between what a human intends and what a formal specification demands. Of 106 tasks, 46 receive zero correct solutions from any tested model.

To close this gap, we propose **SCALPEL** (**S**pecification-**C**onstrained **A**gents **L**earning **P**recise **E**dit **L**ocalization), a reinforcement learning training recipe built on *mutation-consistency rewards*: instead of comparing outputs to a reference answer, we define each task as a pair of counterfactuals. One defines exactly what must change; the other defines exactly what must not. These counterfactuals are executable and require no human labeling. Training with reinforcement learning on these rewards reduces the Unintended Change Rate (UCR) from 1.9% to 1.4%, improves exact solve from 29.2% to 34.0% over the strongest prompting baseline, and reduces mean output length by 65%, indicating the model has learned to edit rather than rewrite.

SVGs are a fully observable, structured domain where every corruption leaves a detectable trace in the element tree, making them a clean proxy for the general problem of side-effect-aware program editing. The reward recipe applies to any domain with a structured, attribute-level representation.

1 Introduction

When you ask a model to edit a structured file, correctness has two independent demands: *do the right thing*, and *do nothing else*. Most benchmarks measure only the first. They reward outputs that look right on the target attribute while remaining blind to anything the model changed that it should not have.

In our experiments, frontier models preserve untouched elements at rates of 95–98%, yet solve only 23–29% of tasks exactly. The dominant reason is a mismatch between human-legible intent and machine-legible specification. A user who writes "make the text black" means any representation of black. A formal specification that requires `#222` will reject `#000000`, even though both are correct to a human eye. The model is not ignorant of the task; it simply expresses the answer in a form the oracle does not accept. That is a specification problem, and no amount of scaling fixes it.

SVGs make this problem concrete and tractable. An SVG is a structured program in which every visual object has a unique ID, every visual property is a typed attribute, and every edit is a localized operation on that attribute tree. When a model rewrites path data it was never asked to touch, the damage is visible in the tree. A pixel-level image metric, which compares rendered output directly, cannot see it at all. This observability is what makes SVGs an ideal testbed: corruption is detectable, ground truth is computable, and the domain is rich enough to surface systematic failure patterns.

Contributions. **SVGym**: 106 SVG editing tasks across 19 fault categories and 4 difficulty tiers, measuring both instruction adherence and preservation. **Quiet corruption analysis**: frontier models preserve untouched elements at 95–98% yet fail

68.9% of tasks; the dominant failure is canonical-value mismatch, not ignorance. **SCALPEL** (Specification-Constrained Agents Learning Precise Edit Localization): a GRPO-based RL training recipe with executable counterfactual rewards that raises exact solve from 29.2% to 34.0% and reduces UCR from 1.9% to 1.4%.

2 Related Work

SVG generation and mutation testing. Existing SVG work scores outputs with CLIP similarity or rasterized FID, metrics that cannot distinguish a surgical edit from a full rewrite (Jia and Harman, 2011; Gulwani et al., 2017). Mutation testing evaluates program correctness by checking whether a test suite can distinguish semantically distinct variants; we apply the same idea to edits, using counterfactual SVGs as the discriminating test.

RL for editing agents. RLHF (Ouyang et al., 2022) and RLAIF (Lee et al., 2023) train editors against learned reward models vulnerable to reward hacking on surface features. GRPO (Shao et al., 2024) eliminates the critic, accepts any scalar reward, and is memory-efficient for 3–7B models, letting us substitute a verifiable mutation-based oracle for a learned judge. SWE-Marathon (SWE-Marathon Authors, 2026) demonstrates the same oracle-first philosophy at benchmark scale, evaluating long-horizon autonomous software agents against verified oracle solutions rather than learned judges; we take inspiration from this and instantiate it at the attribute level through mutation-consistent counterfactuals. Existing benchmarks lack object-level addressable edits, counterfactual labels that separate adherence from collateral damage, and any metric penalizing untouched-element corruption. SVGym provides all three.

3 SVGym

Observation. Each task presents a broken SVG, a natural-language repair instruction, and an object tree that exposes element IDs, attributes, and parent–child structure. The element IDs are the unit of accountability: every changed element in the model's output can be traced back to a specific ID in the input tree.

Task construction. All 106 tasks are built from a 955-icon corpus (Heroicons, Feather, Iconify, and 6 hand-traced company logos) canonicalized to a 128×128px viewBox. Faults are injected programmatically by 11 typed operators. Each task ships with four artifacts: the initial broken SVG, the target SVG, a change record specifying the exact (part, attribute, before, after) changes required, and a preservation set of element IDs that must remain untouched. The preservation set is what makes collateral-damage measurement possible: it names the elements the model had no business touching.

Difficulty tiers and categories. Tasks span four tiers: *Easy* (25): single-attribute faults; *Medium* (35): structural operations on composite scenes; *Hard* (26): compound faults and company logos; *Very hard* (20): generating a missing element from natural-language description. Across tiers, 19 fault categories test distinct competencies (Table 2).

Metrics. **Exact Match (EM)**: string equality after whitespace normalization; strict by design. **Structural Match (SM)**: element tree equality independent of whitespace, isolating cosmetic differences. **Preservation Score (PS)**: fraction of preservation-set elements whose subtrees are unchanged; measures *how much* collateral damage occurred. **Unexpected Parts Rate (UCR)**: fraction of tasks where at least one preserved element was altered; measures *whether* any collateral damage occurred at all. **Expected-Changes Pass Rate (ECPR)**: per required change, whether the model produced the specified (attribute, after-value) pair; the only metric giving partial credit on multi-fault tasks.

4 Mutation-Consistency Rewards

Each task specifies the target edit as a set of acceptable outcomes. We call these **positive mutants** M^+: SVGs that satisfy the instruction, generated by sampling valid values of the target attribute. We also generate **negative mutants** M^-: SVGs that fail the specification by applying a wrong value, damaging a preserved element, or leaving the fault unrepaired. These two sets are the task specification expressed as concrete files rather than prose.

Instruction adherence (r_{adh}) measures how close the model's output is to the positive mutants and how far from the negative ones. We extract the target attribute(s) named in the instruction and

check agreement:

$$r_{\text{adh}} = \frac{1}{|M^+|} \sum_{m \in M^+} \mathbb{1}[A(\hat{S}) \cong A(m)] - \frac{1}{|M^-|} \sum_{m \in M^-} \mathbb{1}[A(\hat{S}) \cong A(m)].$$

The subtracted term is essential. Without it, a model can score perfectly by copying any positive mutant verbatim, including whatever collateral changes that mutant carries. The negative term closes this: any output that matches the attribute fingerprint of a known-bad edit is penalized.

Preservation (r_{pres}) is the fraction of elements in the preservation set that are unchanged in the model's output, equal to the PS metric defined above.

Validity (r_{val}) returns $+1$ if the output is well-formed XML with a root `<svg>` element, -1 otherwise.

The total reward combines all three:

$$r = r_{\text{adh}} + r_{\text{pres}} + 0.5\, r_{\text{val}}.$$

The two main terms have a specific property that makes this reward hard to game. A model that returns the original SVG unchanged maximizes preservation but scores zero on instruction adherence. A model that rewrites the entire SVG to match a positive mutant maximizes adherence but collapses preservation. The only strategy that jointly maximizes both is to make exactly the right change to exactly the right elements, which is the definition of a surgical edit.

5 SCALPEL Training

We train in two phases to address sparse reward signal in early rollouts.

SFT warm-up. We first finetune Qwen2.5-Coder-7B-Instruct on ∼5,000 procedurally generated (broken SVG, instruction, corrected SVG) triples. This gives the model a prior for XML editing before it encounters the RL reward. Without this step, early GRPO rollouts are dominated by malformed XML and the reward carries no useful gradient.

GRPO with a difficulty curriculum. We then train with GRPO (Shao et al., 2024). For each prompt, we sample K=8 candidate outputs, score each with the reward function above, and update the policy to increase the probability of outputs that scored above the group mean. This group-relative advantage estimation requires no critic network: the quality of an edit is judged relative to eight alternative edits for the same task, sampled from the current policy.

To prevent the policy from drifting too far from the SFT initialization, we add KL regularization against the frozen SFT checkpoint (β=0.01). Without this anchor, GRPO's advantage signal can be sharp enough to collapse the policy toward rewarded surface artifacts quickly. QLoRA (Dettmers et al., 2023) (rank 64, α=128) is applied to attention and MLP projections, keeping peak memory manageable for 8-way rollout.

Training follows a four-stage curriculum mirroring SVGym's difficulty tiers, advancing to the next tier only when per-tier mean reward exceeds 0.5. Advancing too early exposes the policy to hard tasks before it has a reliable editing strategy; the threshold ensures basic competence at each tier before the next tier is introduced. The very-hard tier, dominated by element-creation tasks, reaches a plateau of 0.31 mean reward and exhausts its step budget without advancing, confirming that synthesis from natural language remains out of reach for this model size. All other tiers cross the advancement threshold; full hyperparameters and pseudocode are in Appendix E.

Method	EM↑	PS↑	ECPR↑	UCR↓
Gemini-3-Pro	29.2%	98.1%	40.8%	1.9%
GPT-5.5	25.5%	97.9%	37.4%	2.1%
Gemini-Flash	23.6%	95.4%	39.1%	4.6%
Qwen-7B (0-shot)	9.4%	86.3%	17.2%	13.7%
+SFT (ours)	21.7%	96.1%	33.8%	3.9%
SCALPEL (ours)	34.0%	98.6%	49.3%	1.4%

Table 1: SVGym results across 106 tasks. EM = Exact Match; PS = Preservation Score; ECPR = Expected-Changes Pass Rate; UCR = Unintended Change Rate ($1-$PS). Higher is better except UCR. SCALPEL surpasses all prompting baselines on EM while achieving the lowest UCR.

6 Experiments

Prompt baselines. We evaluate three frontier models zero-shot: Gemini 3 Pro Preview, GPT-5.5, and Gemini 3 Flash Preview. All runs are zero-shot at temperature 0; the verbatim system prompt is in Appendix D.

Models are precise editors but poor specification readers. All three frontier models preserve untouched elements at 95–98% (low UCR), yet

exact solve tops out at 29.2%. These two numbers tell the main story: the models are locating the right element, making the right kind of change, and leaving everything else alone. They are just outputting the wrong value. For instance, in all 9 `wrong_color` tasks every model outputs `#000000` or `black` where the spec requires `#222`, and in all 6 `wrong_scale` tasks every model applies a round-number centering heuristic instead of the canonical transform.

7 Analysis

Reward hacking and the value of counterfactual specifications. An early version of the reward used only positive mutants. A model quickly learned to copy a positive mutant verbatim, scoring high on instruction adherence while silently modifying unrelated elements. Adding the negative-mutant repulsive term eliminated this, and the experience points to a general lesson. A specification that says only "this output is good" admits policies that satisfy the letter of the spec through unintended means. A specification that says "this is good and that is bad" is far harder to game, because both constraints must be satisfied simultaneously. The most precise specifications are distinctions between counterfactuals, not reference targets.

UCR as a latent quality axis. Gemini Flash achieves 4.6% UCR; Gemini Pro achieves 1.9%. On a 106-task run, 4.6% UCR means approximately 5 tasks receive silent corruption: an edit that passes inspection on the target attribute while leaving detectable damage in the object tree. UCR is not a secondary metric; it is the metric that matters most when edits accumulate.

8 Conclusion

Frontier models have high preservation and low exact solve because they are precise about *where* to edit but imprecise about *what value* to write. This failure mode is invisible to any evaluation that reports a single accuracy number, and it is not closed by scaling: it is a specification-reading problem.

The open problem is whether surgical agents can be trained to recover specification values, not just specification locations. SCALPEL closes the structural half of that gap. The semantic half remains.

Limitations

SVGym evaluates 2D vector scenes; the reward recipe requires a structured, attribute-level representation and does not transfer directly to unstructured text or natural images. The RL results use a single model family (Qwen2.5-Coder-7B) and a single GPU budget; scaling behavior is left to future work.

References

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *NeurIPS*.

Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program synthesis. *Foundations and Trends in Programming Languages*.

Yue Jia and Mark Harman. 2011. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*.

Harrison Lee and 1 others. 2023. RLAIF vs. RLHF: Scaling reinforcement learning from human feedback with AI feedback. *arXiv preprint arXiv:2309.00267*.

Long Ouyang and 1 others. 2022. Training language models to follow instructions with human feedback. *NeurIPS*.

Zhihong Shao and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

SWE-Marathon Authors. 2026. SWE-Marathon: Can Agents Autonomously Complete Ultra-Long-Horizon Software Work? https://github.com/abundant-ai/long-horizon. Benchmark and evaluation code; preprint forthcoming.

A SVGym Task Taxonomy

Full listing of the 19 fault categories, their difficulty tiers, task counts, and corruption operators used to generate broken SVGs.

Category	Tier	n	Corruption type
`wrong_color`	easy	9	fill/stroke value
`wrong_scale`	easy	6	scale+translate
`wrong_stroke_width`	easy	6	stroke-width
`clipped_viewbox`	easy	4	viewBox shrink
`missing_part`	medium	5	element removal
`extra_part`	medium	5	element injection
`displaced_part`	medium	5	translate offset
`wrong_part_color`	medium	5	fill on sub-element
`duplicate_part`	medium	5	element clone
`flipped_part`	medium	5	reflect transform
`multi_repair`	medium	5	2–3 combined faults
`wrong_color+clipped_viewbox`	hard	6	compound
`wrong_color+wrong_stroke_width`	hard	6	compound
`wrong_scale+wrong_stroke_width`	hard	4	compound
`wrong_scale+clipped_viewbox`	hard	2	compound
`wrong_stroke_width+clipped_viewbox`	hard	1	compound
`wrong_color+wrong_scale`	hard	1	compound
`company_logo_multi_repair`	hard	6	multi on logos
`creation_step`	very_hard	20	synthesize absent part
Total		**106**	

Table 2: SVGym fault categories. Compound categories require simultaneously repairing two independent faults; `creation_step` requires generating a new SVG element from natural-language description alone.

B Per-Category Results

Exact-match breakdown by category for all three frontier models, sorted by descending solve rate.

Category	n	Pro EM	GPT-5.5 EM	Flash EM
`wrong_stroke_width`	6	100%	100%	100%
`extra_part`	5	100%	100%	100%
`duplicate_part`	5	100%	100%	100%
`wrong_stroke_width+clipped_viewbox`	1	100%	0%	0%
`displaced_part`	5	80%	80%	40%
`flipped_part`	5	80%	60%	40%
`wrong_part_color`	5	60%	60%	60%
`clipped_viewbox`	4	50%	0%	0%
`creation_step`	20	5%	5%	5%
`multi_repair`	5	0%	0%	20%
`wrong_color`	9	0%	0%	0%
`wrong_scale`	6	0%	0%	0%
`missing_part`	5	0%	0%	0%
`wrong_color+clipped_viewbox`	6	0%	0%	0%
`wrong_color+wrong_stroke_width`	6	0%	0%	0%
`wrong_scale+wrong_stroke_width`	4	0%	0%	0%
`wrong_scale+clipped_viewbox`	2	0%	0%	0%
`wrong_color+wrong_scale`	1	0%	0%	0%
`company_logo_multi_repair`	6	0%	0%	0%

Table 3: Exact match by category. “Pro” = Gemini 3 Pro Preview; “Flash” = Gemini 3 Flash Preview. 46 tasks receive zero exact solutions from any model.

C ECPR by Attribute

Attribute	Frontier avg.
stroke-width	83%
cy	83%
exists	75%
points	67%
fill (part)	52%
cx	44%
x (transform)	15%
viewBox	10%
y (transform)	7%
size (scale)	2%
color (fill)	0%

Table 4: ECPR by attribute. "Frontier avg." is the mean pass rate across Gemini 3 Pro, GPT-5.5, and Gemini Flash, aggregated over all models and tasks. The hardest semantic attributes (color, transform coordinates, and viewBox) achieve near-zero pass rates, confirming that frontier models fail on specification precision rather than structural understanding.

D Evaluation Reproducibility

System prompt (verbatim). *"You are a precise SVG editor. Return ONLY the corrected SVG as a complete, valid document. No markdown fences, no commentary."*

Model identifiers. Gemini 3 Pro Preview: `gemini/gemini-3-pro-preview` via LiteLLM. GPT-5.5: `gpt-5.5` via OpenAI API. Gemini 3 Flash Preview: `gemini/gemini-3-flash-preview` via LiteLLM. All runs: temperature 0, single pass, 2026-05-26.

Latency. Mean per-task: Gemini Pro 9.95s, GPT-5.5 9.08s, Gemini Flash 2.90s. Gemini Flash's 3.4× speed advantage over Pro yields only a −5.6pp exact-match penalty, making it the most efficient frontier option.

Dataset and code. Tasks: `data/tasks/*.json` (106 files). SDK: `sdk/python/` (pip-installable; includes oracle and noop solvers). Benchmark harness: `scripts/benchmark-litellm.py`. Live viewer: anonymized for review.

E Training Hyperparameters and Pseudocode

Hyperparameters. Table 5 lists all values used in the reported runs.

Hyperparameter	Value
Base model	Qwen2.5-Coder-7B-Instruct
SFT training examples	∼5,000 procedurally generated triples
GRPO rollouts per prompt (K)	8
KL penalty coefficient (β)	0.01
QLoRA rank / α	64 / 128
QLoRA target modules	attention + MLP projections
Curriculum threshold	mean reward > 0.5
Steps per stage (easy / medium / hard)	17 / 34 / 51
Steps for very_hard stage	50 (budget; threshold not reached)
Total GRPO steps	152
Independent seeds	3
Approx. wall-clock time	∼1.9 hours (GPU)

Table 5: SCALPEL training hyperparameters.

GRPO training loop. Algorithm 1 gives the pseudocode for the four-stage curriculum GRPO loop used by SCALPEL. The policy is initialized from the SFT checkpoint before the loop begins.

Algorithm 1 SCALPEL: Curriculum GRPO Training

Require: SFT policy π_θ; stages $\mathscr{S}$; reward r; K=8; β=0.01; τ=0.5

```
for each stage s ∈ 𝒮 do
    repeat
        Sample prompt x from stage-s training tasks
        Sample K outputs {o_1, ..., o_K} ~ π_θ(· | x)
        Score each: r_i ← r(o_i, x)
        Compute advantage: Â_i ← (r_i − r̄) / (σ_r + ε)
        Update θ: maximize E[Â_i log π_θ(o_i|x)] − β D_KL(π_θ || π_SFT)
    until stage mean reward > τ or step budget exhausted
end for
```

Logged metrics. Per-step metrics: policy loss, mean reward, reward std, fraction of zero-advantage groups, mean output tokens, curriculum stage. Every 50 steps the model is evaluated on the dev split: structural match, preservation score, token ratio, and pass rate by difficulty tier.

F Benchmark Visualisations

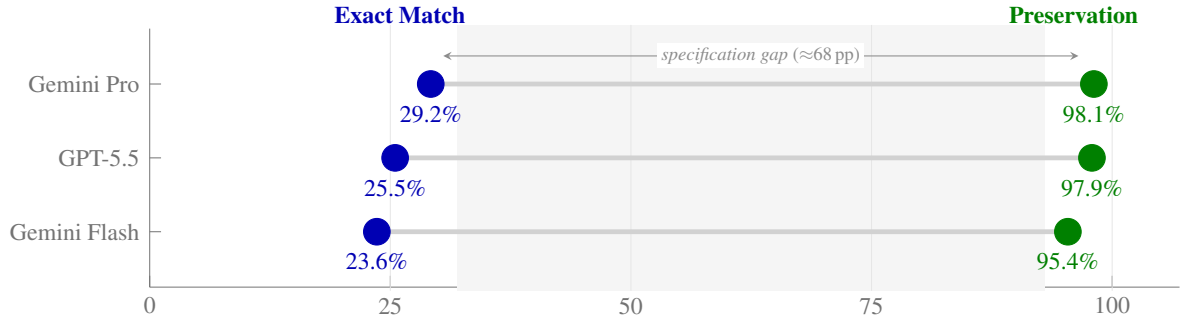

Figure 1: **Frontier models preserve what they should not touch, but cannot recover the exact specification.** Each row is one model. **Exact Match**: fraction of tasks where the model output the canonical corrected SVG. **Preservation**: fraction of untouched elements left unchanged. All models score ≥95% on preservation yet no model exceeds 30% exact match. The gap arises because models output semantically equivalent but canonically wrong values (e.g. `black` instead of `#222`): they know *what* to change but not *exactly how*.

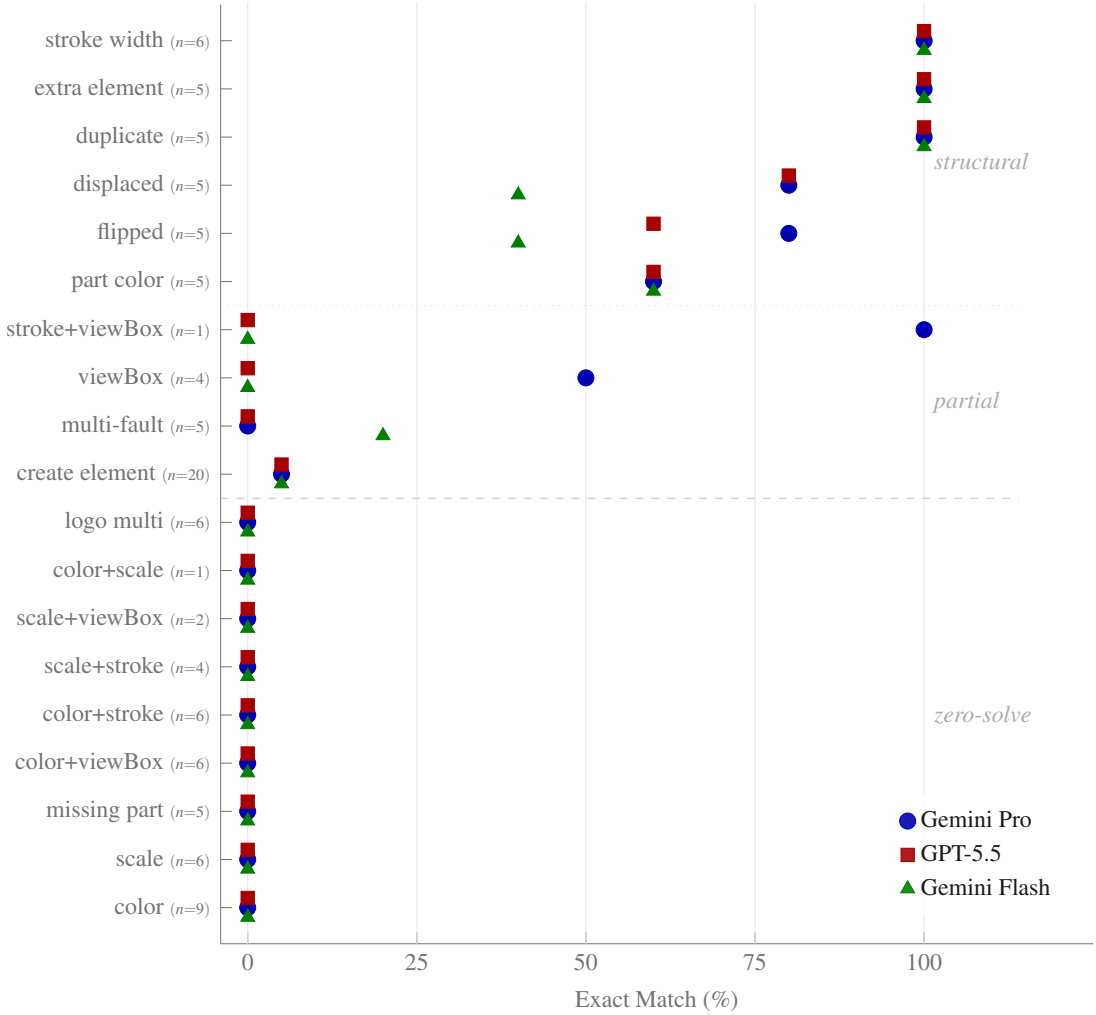

Figure 2: **Frontier models solve structural edits reliably but fail on semantic and compound tasks.** Categories sorted by mean exact match (descending). *Structural* tasks (remove/add/move an element, fix stroke width) cluster at 100% for all models (top, above dotted line). *Semantic* tasks requiring exact value recovery (color, scale, viewBox) and *compound* tasks (two simultaneous faults) fall into the zero-solve zone (below dashed line): 9 of 19 categories receive zero exact solutions from any model.

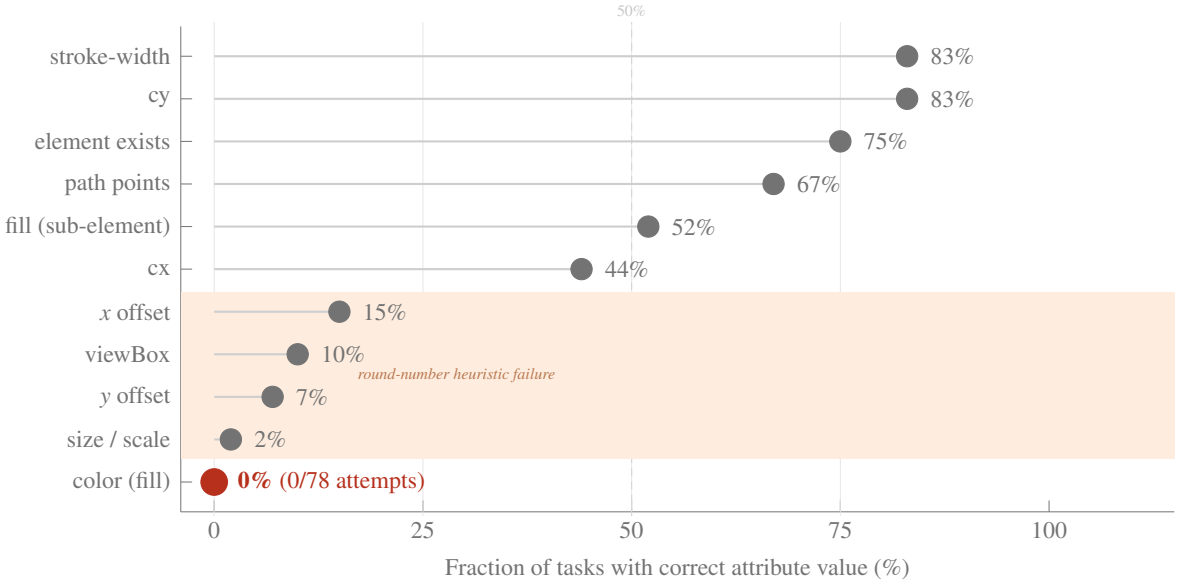

Figure 3: **Color is the hardest attribute: 0 correct answers out of 78 attempts.** Each dot shows the fraction of tasks where the model produced the exact canonical attribute value, aggregated across all models and tasks. The shaded band marks the *round-number heuristic failure*: *x*/*y*/size offsets and viewBox coordinates score below 15% because models apply nearest-integer approximations rather than recovering exact canonical transforms.

	Gemini Pro	GPT-5.5	Gemini Flash
Easy n=25	32%	24%	24%
Medium n=35	**60%**	**57%**	**51%**
Hard n=26	4%	0%	0%
V. Hard n=20	5%	5%	5%

performance cliff

EM: 0% high

Figure 4: **Solve rate collapses from Medium to Hard: structural operations dominate the tractable tier.** Cell intensity encodes exact-match rate (white = 0%, dark blue = high). Medium tasks peak at 51–60% because the tier is dominated by structural operations (add/remove/move elements). Compound faults in Hard and synthesis in Very Hard flatten solve rates to near-zero for all models. The single non-zero Hard cell (Gemini Pro, 4%) corresponds to a single-task category (n=1, `wrong_stroke_width+clipped_viewbox`) whose two individual fault types are both reliably handled by Gemini Pro in isolation.