TIDYUP: POST-ALIGNMENT OF AGENTIC LAYOUT OUTPUTS WITH A TINY SPECIALIST

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language and vision–language agents can now assemble and edit real graphic designs, but they reason about geometry imprecisely: leaving overlaps, broken alignments, uneven spacing, and off-canvas content. The prevailing fix—re-invoking the same or a larger model to critique a rendered canvas and rewrite coordinates—is slow, costly, non-deterministic, and prone to regressing already-correct parts of the design. We argue that this narrow, geometric defect class does not call for more generalist reasoning but for a *specialist*. We present *TidyUp*, a tiny (∼86M-parameter), vision-free pipeline that repairs a design in place after the agent has already done the visual reasoning. A lightweight *reflection router* first flags which elements are broken and by what violation. Specialist *correctors* — first to *jointly condition* intricate design *group structures* and prior *design state* — apply hierarchical attention over local and global alignment geometry to nudge flagged elements into place, leaving size, type, and unedited elements untouched, with *no regeneration*. Unlike prior flow-based layout models, trained on noise-to-data, each corrector is trained for *refinement*, integrating its ODE from the broken layout. Owing to varying repair severity, TidyUp *iterates* reflection-correction with an auto-stop. Correctness-preserving guards—including a violation-regression bailout—guaranties TidyUp never returns a layout worse than its input. TidyUp, trained on AI-edited designs from a *large-scale* production system and a public *Crello* benchmark, with synthetic defects derived from multi-turn editing patterns, matches or exceeds the SOTA-level performance at a fraction of the cost: $1.2s$ latency at ∼250 RPM on a single H100, ∼160 MB footprint, with no additional agent calls.

1 INTRODUCTION

Generative agents built from large language models (LLMs) and large vision–language models (LVLMs) increasingly produce and edit real graphic designs—posters, slides, social creatives, app screens, by emitting or rewriting structured descriptions of a canvas (Tang et al., 2024; Li et al., 2026). These agents are strong at the *semantic* decisions of design: what elements belong on the canvas, what they say, and roughly where they go. But they are conspicuously weak at the *geometric* execution of those decisions as the reasoning happens in the token or JSON space rather than in continuous pixel space. As a result the outputs systematically exhibit *a small*, recurring family of defects: elements that overlap when they should not, edges that almost-but-not-quite align, uneven gaps, and content that bleeds off the canvas (Li et al., 2026).

In practice agents rarely start from a blank canvas but instead *edit* an *existing design* over many turns, moving, adding, resizing a handful of elements at a time while leaving many untouched. This gives rise to compounding effects: an element added early collides with one moved later, and a caption nudged in one turn drifts out of alignment with a title fixed in another (Laban et al., 2026). In our analysis of 724 editing sessions on a *production system* over a period of 9 weeks, 34.6% of issues were due to at least one overlap or alignment violation by the 4th editing turn (Figure 14) of the agent. The result is a *locally damaged* but mostly-correct layout, not the one that needs a *rethought*.

Figure 1a shows an example of how layout breaks over the course of ordinary multi-turn edits in the *production system*. How should then such defects be fixed? The dominant answer today is "self-correction": re-invoking a large model to inspect its own output, through prompting a vision-language model to critique the canvas (Zhang et al., 2024; Jang et al., 2026), or by expensive fine-

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

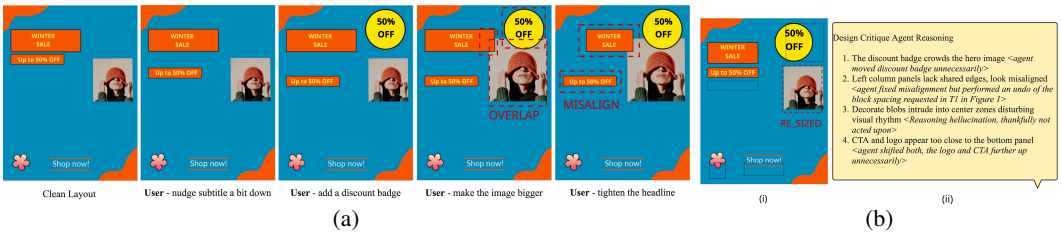

Figure 1: (a) A design edited over four ordinary agentic turns: cross-turn compounding introduces two violations, an overlap at T_3 and a misalignment at T_4. (b) A large LLM used as a critic to fix the resulting layout issues (overlap/alignment/spacing): it fixes one problem but introduces new ones.

tuning of the agent itself to reason spatially (Li et al., 2026) or more generally Kumar et al., 2025. Both inherit the generalist's cost and unreliability at exactly the wrong moment—slow and expensive to call, and often unable to reliably correct geometric mistakes without external ground-truth feedback (Huang et al., 2024). Figure 1b illustrates a fix of the layout broken in Figure 1a leveraging GPT-5 (OpenAI, 2025) via design reflection (overlap/Alignment/Spacing). It fixes the alignment but unnecessarily moves three elements, undoing an explicit user action from turn $T3$.

A separate line of work already encodes a strong prior over well-formed layouts by training flow- and diffusion-based *generative* models (Guerreiro et al., 2024; Chen et al., 2024; Iwai et al., 2024) to produce a good layout from noise. None of these transfer to the agentic setting directly: they have no notion of a previous design state to preserve, and left to run on a broken input, they would rearrange the whole canvas rather than mend only what a later edit damaged. Likewise, the existing corruption processes either start from noise or perturb a clean layout with random jitter, and known metrics — FID, raw overlap/alignment, assume a simplicity real production designs (variable depth, opacity, creativity-driven sub-groups) don't have only measuring distribution similarity suited for regeneration rather than a focused repair.

Our central claim is that the agentic era calls for a division of labor. The agent, having already reasoned—often with vision—about *what* to make, should not be asked to also perfect the geometry; a small, cheap, specialized model should clean up after it. This re-frames the problem as *post-alignment of agentic outputs*, analogous to how a lightweight verifier is increasingly paired with a large generalist elsewhere (Pan et al., 2024; Zhong et al., 2024).

We realize this *division of labor* in **TidyUp**, a tiny, vision-free pipeline that post-aligns the output of any design agent. TidyUp proposes an *reflection router* which localizes and classifies each broken element and dispatches it to one of two *correctors* built on *LayoutFlow* Guerreiro et al., 2024 flow-matching architecture, here constrained to *repair* rather than generation. Due to variable repair intensity, TidyUp iterates the reflect-repair cycle until the guardrail mechanism confirms no violation or a regression thus avoiding making things worse. Its reliable delivery rests on four **contributions:**

Post-alignment by a tiny, non-reasoning specialist: We cast the cleanup of agentic design errors as *post-alignment* and solve it with TidyUp, a compact, vision-free model that detects and repairs geometric defects without any reasoning, under correctness-preserving guards which guarantee that no output is ever worse than its input.

A high-recall reflection router: The pipeline is steered by an error router that flags, at high recall, which elements are broken and by what violation type, dispatching each flagged element to its specialized corrector in an iterative loop.

Specialized flow-matching correctors: Each error specific corrector steers a pretrained flow-matching generator over a bounded interval from the broken layout, not from scratch. Each contributes target-specific supervised losses, refinement-mask–conditioned attention (leveraging but not blindly trusting, the router), state conditioning that soft anchors what's unchanged yet lets the neighbor move if needed, and user-group–aware relational attention that keeps grouped elements coherent.

Distribution-matched corruption and depth/group-aware metrics: We synthesize corruptions matched to a real production assistant's defect distribution, over designs with variable depth and creative element sub-groups — complexity no existing data benchmarks capture — and revisit the standard metric suite (FID, overlap/alignment) to measure repair, not regeneration. We validate on

production system designs and Crello Yamaguchi, 2021 benchmark, outperforming prior generative and self-correction baselines.

2 Related Work

Layout generation as a prior over well-formed designs: Discrete and continuous diffusion models (Chen et al., 2024; Iwai et al., 2024; Mayet et al., 2025) and flow-matching generators (Guerreiro et al., 2024; Shaul et al., 2025) produce element boxes and categories that respect implicit design conventions, and thus encode exactly the prior over "what a good layout looks like" that a corrector needs. However, they are trained to synthesize layouts from noise or partial specifications, not to repair a specific corrupted input; devoid of depth ordered elements, user-defined element groups.

Self-correction by the generalist: Recent systems cast layout as a language, prompting an LLM to emit a structured description (e.g., HTML/JSON) of the canvas (Tang et al., 2024) though good at semantic composition they reason about geometry symbolically, exhibiting systematic spatial defects.The dominant remedy is to invoke a powerful model again. Training-free approaches render the layout to an image and let an LVLM critique and rewrite coordinates over several rounds (Zhang et al., 2024; Jang et al., 2026); reinforcement-learning approaches instead fine-tune the agent's own spatial reasoning (Li et al., 2026). Both inherit cost, latency and chances of regressing elements that were already correct. Research strongly contends that self-correction is exactly where large models are weakest and warrant an external feedback (Huang et al., 2024). Agent's reliability decays over multi-turn interaction (Laban et al., 2026), and often warrants an expensive dedicated training of the generalist (Kumar et al., 2025). TidyUp instead offloads correction to a small specialist, consistent with the broader finding advocating post-hoc corrections instead (Zhong et al., 2024).

Optimization-based post-processing: bypass the large model, adjusting geometry directly via constrained optimization—alternating discrete assignment and continuous refinement to remove layout defects (Shen et al., 2025). (Chen et al., 2024) applies its aesthetic constraints as a post-processing step. But they fail selecting among many geometrically valid but aesthetically different fixes.

Error localization: Closest to our router, Layout-Corrector (Iwai et al., 2024) and Spot the Error (Lin et al., 2024) learn to locate erroneous tokens *within* a layout generator and re-predict them. Our router instead operates vision-free and post-hoc and repairs rather than regenerates (section 4.3).

3 Problem Formulation

Layouts and agent edits: We represent a design as an ordered set of N elements $\mathcal{L} = \{e_i\}_{i=1}^{N}$. Each element $e_i = (b_i, c_i, o_i, s_i, g_i)$ has a bounding box $b_i = (x_i, y_i, w_i, h_i)$ normalized to the canvas, a category c_i (text, image, shape, *user-group* container, etc.), a stacking order o_i, a subtype s_i, and a group pointer g_i to its parent container ($\varnothing$ if top-level). TidyUp is never shown the rendered image as the content-aware judgments have been already made by the agent. An editing agent transforms a clean prior layout $\mathcal{L}^0$ into a new candidate layout $\tilde{\mathcal{L}}$ by moving, adding, or removing elements. The elements it moves or adds form the *changed* set $\mathcal{C} \subseteq \{1, \ldots, N\}$, while the remaining elements are left as they were. $\mathcal{C}$ is exposed to the corrector network as previous-state information.

Defects: The candidate $\tilde{\mathcal{L}}$ contains geometric defects on a subset of *broken* elements $\mathcal{B} \subseteq \{1, \ldots, N\}$, each with a type in $\{\text{OVERLAP}, \text{ALIGNMENT}\}$ (spacing and off-canvas violations are handled as special cases of these two). These are defects of *placement*, not of content or size: the editing agent chose the right elements but positioned them imprecisely.

Task and the position-only invariant: Given $\tilde{\mathcal{L}}$ and $\mathcal{C}$, the corrector must produce $\mathcal{L}^*$ that removes defects on $\mathcal{B}$ and as far as possible preserve the non-defective structure of $\tilde{\mathcal{L}}$.TidyUp enforces one *hard* invariant: it can only *translate* element's position (x,y), keeping its size (w,h) and other attributes fixed,

$$\mathcal{L}^* = \{(x_i^*, y_i^*, w_i, h_i, c_i, o_i, s_i, g_i)\}_{i=1}^{N} \tag{1}$$

so by construction it can never resize, re-type, or reorder an element—a guarantee that is hard to obtain from a free-form generative rewrite. *Which* elements move, by contrast, is a *soft* preference rather than a constraint: the router and previous-state conditioning steer the model to reposition mainly the flagged elements and leave the rest in place, but the corrector keeps the freedom to nudge an unflagged neighbor when a violating element has nowhere clean to go—we deliberately trust the corrector's learned design-space prior over any per-element mask from the router.

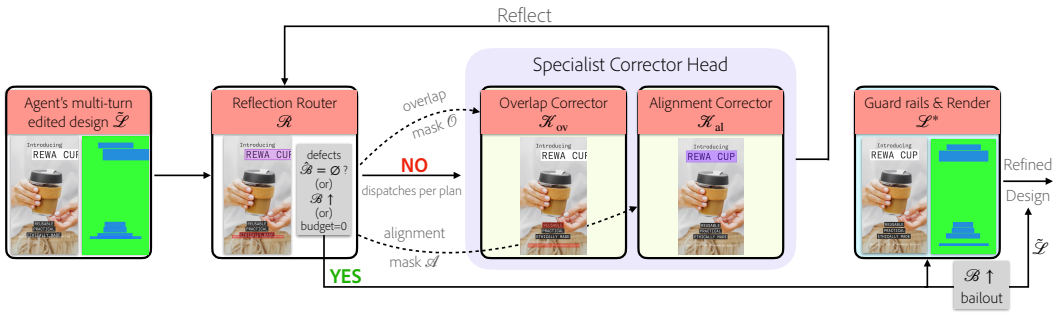

Figure 2: The TidyUp pipeline. An agent returns an edited design with geometric defects (left). Each round, the router flags which elements are broken and classifies each as an overlap (red overlay) or alignment (purple overlay) violation; flagged elements are routed to the matching specialist corrector, which repairs them by translating only those elements. Correctness-preserving guards and a violation-regression bailout wrap the loop so the output is never worse than the input (right). Every component operates on the wireframe—boxes and categorical attributes—with no rendering.

4 METHOD

TidyUp comprises three learned components (Fig. 2): a reflection router $\mathcal{R}$ and two specialist correctors—an overlap corrector $\mathcal{K}_{\text{ov}}$ and an alignment corrector $\mathcal{K}_{\text{al}}$. All three read only the *wireframe*—each element's box b_i and categorical attributes (c_i, o_i, s_i), with no image ever rendered.

4.1 BACKGROUND: FLOW MATCHING

Flow matching learns a continuous-time transport from a base distribution to the data distribution. For a clean sample x_1 and a base sample $x_0 \sim \mathcal{N}(0, I)$, the linear (optimal-transport) path $x_t = (1-t)\,x_0 + t\,x_1$ interpolates between them over $t \in [0, 1]$—with $t{=}0$ noise and $t{=}1$ data (Guerreiro et al., 2024) and has the constant velocity $x_1 - x_0$, which a network $v_\theta(x_t, t)$ is trained to regress:

$$\mathcal{L}_{\text{fm}} = \mathbb{E}_{t,\,x_0,\,x_1} \left\| v_\theta(x_t, t) - (x_1 - x_0) \right\|^2 \tag{2}$$

New samples are produced by integrating the ODE $\dot{x} = v_\theta(x, t)$ from x_0 at $t{=}0$ to $t{=}1$. Our correctors build on LayoutFlow (Guerreiro et al., 2024), which instantiates this recipe for layouts by representing each element's box and category as a token. We expand that per-element token with additional categorical attributes—subtype, stacking order, and group membership—and adapt the backbone from generation to *repair* in what follows.

4.2 SPECIALIST CORRECTORS

A corrector takes the agent's broken layout $\tilde{\mathcal{L}}$ and decides *where* each broken element (the unknown set $\mathcal{B}$) should move — an under-determined task, as towards resolving the issue many placements could be geometrically legal. We repair instead with a learned *generative prior* over well-formed layouts: the flow-matching generator of section 4.1, is adapted to take an editing agent's *broken output* as the input and produce a directed coordinate edit rather than free generation.

Why two correctors: Agentic edit produced defects overlap and their fixes span a large, tangled region of layout space. Our tiny model struggles to learn a single prior that covers all of them at once. We split correction into two *specialists*—an overlap corrector $\mathcal{K}_{\text{ov}}$ and an alignment corrector $\mathcal{K}_{\text{al}}$, sharing the architecture and repair mechanism, differing in training data and objectives.

Directed, coordinate-only edits: We first make the edit directed. Only (x, y) channels are integrated, every other attribute (Eqn. 2) passes through unchanged, so a corrector may reposition but never resize or re-type. Because layout is a *set* rather than a sequence, positional encodings index each element's stacking order (depth) rather than sequence position, making the corrector permutation-equivariant yet depth-aware.

Learning to repair, not generate: We then teach the generator to start from broken layouts alongside the clean designs used for ordinary generation. We synthesize *broken* samples matching a real editing agent's defect distribution (see Appendix A.2), with each training step drawing a fixed 3:1 mixture: 75% of sample integrated from noise (generation) and remaining 25% from the element's broken state (repair). A broken layout $\tilde{x}$, already close to its clean counterpart x_1, is projected onto

the flow near the clean end, at t_{low} rather than treated as noise at $t{=}0$.

$$x_{t_{\text{low}}} = \tilde{x}, \qquad x_t = \tilde{x} + \tfrac{t-t_{\text{low}}}{1-t_{\text{low}}}\,(x_1 - \tilde{x}), \quad t \in [t_{\text{low}}, 1], \tag{3}$$

with v_θ trained toward x_1 over this short late interval and integrated from t_{low} to $t{=}1$ with a few Euler steps at inference. A repair-only training would collapse the velocity field towards near-zero (Appendix A.6.1); interleaving clean samples prevents this. Starting the uncontrolled field from $\tilde{x}$ at t_{low} trades defect removal for input preservation. This bounded-interval repair admits an optimal-control view of the pretrained flow (Wang et al., 2025).

Group-aware correction: Real designs are not flat: their elements form semantically coherent groups (a logo lockup, a captioned image a button and its label). Repairing an element in isolation can tear such a group apart, so a corrector must move a group's members coherently. Each transformer layer composes four masked-attention passes- sibling, parent, top-level and global (more details in Appendix A.6.2):

$$\begin{aligned} &\text{Attn}_{M^{(\ell)}}(H) = \text{softmax}\Big(\tfrac{(HW_Q)(HW_K)^\top}{\sqrt{d}} + M^{(\ell)}\Big)HW_V, \\ &H \leftarrow H + \text{Attn}_{M^{(\ell)}}(H) \quad \text{for } \ell \in (\text{sib, par, top, glob}). \end{aligned} \tag{4}$$

where the level mask $M_{ij}^{(\ell)}{=}0$ on the level-ℓ edges—siblings, container-to-children, and top-level respectively, and all pairs for the global level—and $-\infty$ elsewhere. The first three levels attend by the grouping arrangement, so information settles within a group before the final global pass mixes it across the canvas. This mirrors the local/global attention of graph transformers (Veličković et al., 2018; Dwivedi et al., 2026) and the intra-/cross-table attention of relational transformers (Ranjan et al., 2026), applied here to a design's group hierarchy.

Architecture: Both specialists share a small, vision-free backbone: an encoder-only four layers transformer ($d_{\text{model}}{=}512$, 8 heads) in which the flow time t is injected through adaptive layer normalisation. Each element is a single token whose box (x, y, w, h) and categorical attributes are encoded as analog bits into an 18-dimensional output, keeping each corrector at $\sim$36M parameters.

4.3 Reflection Router

Why a router: The *reflection router* $\mathcal{R}$ supplies a selectivity on which element in a design is broken and by what problem. Without which an element if handed to both the correctors, they would interfere, where one would undo other's fix. It's backbone follows (Iwai et al., 2024)— re-purposed from a component *inside* a generator into a standalone, post-hoc detector with nuanced contributions.

Discrete by design: Unlike the correctors' continuous coordinate space, the router works over quantized tokens by design – we learn that binning collapses sub-bin jitter, so only a meaningful displacement changes an element's tokens; a continuous-space router does not match this performance due to its hypersensitivity (Appendix A.6.3).

Break-type head and routing: A linear head maps each element's representation h_i to a distribution p_i over the three break types $\{\text{NONE}, \text{OVERLAP}, \text{ALIGNMENT}\}$; a single head thus performs detection, violation typing, and routing at once: with $\hat{y}_i = \text{argmax}_k\, p_{i,k}$, the flagged set $\hat{\mathcal{B}} = \{i : \hat{y}_i \neq \text{NONE}\}$ splits into an overlap set $\mathcal{O} = \{i : \hat{y}_i = \text{OVERLAP}\}$ and an alignment set $\mathcal{A} = \{i : \hat{y}_i = \text{ALIGNMENT}\}$, one per corrector.

Training for high recall: Router is tuned towards high *recall* via explicit down-weighing of clean and easy samples and re-arrangement of the element embedding space using supervised InfoNCE term (Rusak et al., 2025), as the defect it misses will be ignored by the correctors. False positives, on the contrary, are comparatively harmless — a correctly placed element already lies near the design manifold, so the repair flow barely moves it. The router has $\sim$14M trainable parameters.

Router conditioning: To let the router reason in the space of *well-formed* designs, the encoder that produces h_i is conditioned on a separate, frozen flow-matching generator—a generation-only model, distinct from the repair-finetuned correctors. We run this generator for a single step near the clean end of the flow ($t_0{=}0.9$; recall that $t{=}1$ is data) on the broken layout and capture the hidden states of its last four transformer layers. Each of the router's four transformer layers then cross-attend its per-element tokens z to the corresponding one of these four generator states,

$$\{g^{(\ell)}\}_{\ell=1}^{4} = \text{FlowEnc}_{t_0}(\tilde{\mathcal{L}}) \;\; \text{(frozen)}, \qquad h_i = \text{Enc}\big(z;\, \{g^{(\ell)}\}\big)_i \tag{5}$$

where the router's ℓ-th layer attends to the generator state $g^{(\ell)}$, so the router inherits the generator's learned geometry prior while staying entirely in wireframe space. Generator being trained in *continuous* space, feeds the router fine-grained geometric information that the quantized tokens lack.

Corrector conditioning: Router provides a per-element mask m to the respective corrector model, as the corrector's learned prior is reliable only in regions of the design manifold that are well represented in its training data. The mask m confines a pass to a non-overlapping subset of violating elements, keeping every correction a short, well-supported step along the manifold, while the iterative router loop supplies these subsets one at a time. Mask is injected in two ways: (1) every self-attention layer is conditioned on the mask so that an unbroken element cannot attend to broken-element keys and the clean anchoring context stays uncontaminated; (2) in the objective, repair samples are up-weighed. Finally to reduce sensitivity of router's false positives, a small fraction of mask entries are randomly flipped during training, increasing corrector's resiliency.

4.4 EDIT-AWARE CONDITIONING

Design edited several time in multi-turn edits offers a signal on which elements the agent changed (with no access to their previous coordinates). This helps the corrector repair the geometric damage of an edit without regressing the design to what the prior deems typical. This conditioning is added without retraining from scratch. First the base corrector is trained as discussed and then a lightweight state-consumption module is attached—which is finetuned on state-augmented data leveraging whether an element was changed by the agent. Similar to how a text-to-image diffusion model is adapted into an inpainting model: a capable generator is taught to honour extra conditioning rather than re-learn from zero. The new module is *zero-initialized*, so at the start of finetuning the state-conditioned model is byte-for-byte the base corrector and degrades nothing; it then learns to use the edit flags as the finetuning proceeds. The edit signal $s = \{\mathbf{1}[i \in \mathcal{C}]\}_i$—enters as a soft prior, applied at inference via classifier-free guidance at strength w,

$$\tilde{v}_\theta = v_\theta(x_t, t, c) + w\big(v_\theta(x_t, t, c, s) - v_\theta(x_t, t, c)\big), \tag{6}$$

so $w{=}0$ recovers the unconditioned corrector and larger w concentrates edits on what the agent changed. Being soft, the model remains free to move an unchanged neighbour when that's the only way to place a changed element well—it stops disturbing settled parts of the design gratuitously.

4.5 TRAINING OBJECTIVES

The field is supervised by a weighted flow-matching loss. In order to obtain frequency-balanced weights w_i so that the correct elements neither dominate the loss nor drop out of it each broken element is assigned a raw weight of 1 when flagged by the refinement mask m and α otherwise ($\alpha{=}0.5$), and are normalized to sum to one per layout where the edit signal s is supplied on enabling the edit-aware conditioning.

$$\mathcal{L}_{\text{flow}} = \mathbb{E}_{t,\,x_0,\,x_1} \sum_i w_i \left\| v_\theta(x_t, t, c, m, s)_i - (x_1 - x_0)_i \right\|^2 \tag{7}$$

The same per-element weighting w_i carries over to a light position L_1 term $\mathcal{L}_{L_1}$ and to a boundary loss $\mathcal{L}_{\text{bd}}$ that keeps boxes inside the canvas. Alignment, is poorly captured by a reconstruction loss as two layouts with equal coordinate error differ sharply in whether edges line up. We add a *global alignment loss* on the reconstructed clean estimate $\hat{x}_1 = x_t + (1-t)\, v_\theta$. From the clean training layout we read off its alignment graph $\mathcal{G}$—the anchor pairs (i, j, a), with a one of the six edges or centers (left/center/right in x, top/center/bottom in y), that coincide within a tolerance τ and penalize the residual of exactly those pairs on $\hat{x}_1$, weighted toward late t where the estimate is reliable,

$$\mathcal{L}_{\text{align}} = \mathbb{E}_t \left[\omega(t) \sum_{(i,j,a)\in\mathcal{G}} \big|\, \phi_a(\hat{x}_1^{\,i}) - \phi_a(\hat{x}_1^{\,j}) \,\big| \right], \qquad \omega(t) = e^{\,k(t-1)} \tag{8}$$

where ϕ_a returns anchor a's coordinate. Unlike LACE's Chen et al., 2024 unsupervised "snap-everything" prior, $\mathcal{G}$ is read from the design's *own* structure, so the loss restores the alignments the layout was meant to have rather than inventing new ones. The four terms combine as:

$$\mathcal{L}_{\text{total}} = \gamma_{\text{flow}}\, \mathcal{L}_{\text{flow}} + \gamma_1\, \mathcal{L}_{L_1} + \gamma_b\, \mathcal{L}_{\text{bd}} + \gamma_{\text{align}}\, \mathcal{L}_{\text{align}} \tag{9}$$

and the two correctors differ only in which terms are active. The **overlap corrector** $\mathcal{K}_{\text{ov}}$ turns off alignment (γ_{align}=0), training on flow, boundary, and L_1 terms and may shift an unflagged neighbor to open space for a colliding element. The **alignment corrector** $\mathcal{K}_{\text{al}}$ switches flow term off (γ_{flow}=0) and trains on alignment loss (with boundary and L_1), nudging the flagged elements until their edges and centers line up with the rest of the design.

4.6 A DO-NO-HARM INFERENCE CALCULUS

At inference, the router inspects the current layout —left by the previous corrector along with the original input from the agent and flags the broken elements and their types, and dispatches the overlap and alignment sets to their specialists; this repeats in a closed loop until the router flags nothing or a small round budget is reached. We believe that detection-repair looping isn't novel but dispatching the issues selectively is particular to TidyUp, so is with the guarantee of *do-no-harm*: we repair same broken input rather than generate one, the result is comparable against what it started with. Let $V(\mathcal{L})$ be the estimated violation count the router reports for a layout; a round producing $\mathcal{L}_{\text{corr}}$ from input $\tilde{\mathcal{L}}$ is kept only if it does not increase that count,

$$\mathcal{L}^\star = \begin{cases} \mathcal{L}_{\text{corr}}, & V(\mathcal{L}_{\text{corr}}) \le V(\tilde{\mathcal{L}}), \\ \tilde{\mathcal{L}}, & \text{otherwise}, \end{cases} \qquad \Longrightarrow \qquad V(\mathcal{L}^\star) \le V(\tilde{\mathcal{L}}). \tag{10}$$

Therefore TidyUp offers a safety in deployment- it never returns a layout worse than the one the agent produced—a property, a from-scratch generator cannot offer. On an average we see that iterations do 3 rounds taking $1.2s$ on a single H100 with no calls to the design agent.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Data: The production system designs' element schema exposes: per-element opacity, depth-ordered (z-indexed) stacking, and explicit element groups. Among the standard layout benchmarks—Crello (Yamaguchi, 2021), RICO (Deka et al., 2017), PKU PosterLayout (Hsu et al., 2023), CGL (Zhou et al., 2022), and PubLayNet (Zhong et al., 2019)—Crello is the only one that does show complexity nearer to these production designs (dataset comparison Appendix A.1. We therefore evaluate on Crello, using its standard benchmark split (Train : 19421 / Validation : 1875 / Test : 1955) (23,251 vector design templates, at most 50 elements per design).

We use an internal corpus of ∼0.5M clean design documents, randomly partitioned into train/validation/test splits (details in Appendix A.3). Crucially, it exercises *user-grouped containers* at scale—the grouping primitive that is ubiquitous in today's design and presentation tools (PowerPoint, Canva, Adobe Express, Figma etc). From both corpora we synthesize three broken-layout datasets—one for each of the two correctors and one for the router—each generated per split, so no design leaks across the partition. The validation split records the models' improving capabilities across ablations in Table 1, while the held-out test split drives the competitor comparison of Table 2.

Training: Each corrector is trained in the two stages: a base stage and a state-conditioned finetuning stage, each for 600K iterations at batch size 2048 and learning rate $5\text{e}{-4}$ on 2×H100 (∼1 day per stage). Stage 2 enables edit-aware state conditioning—whose newly added state-processing modules use a $10\times$ larger learning rate—while all other hyperparameters carry over from Stage 1. The router is likewise trained in two stages: its frozen base flow generator is trained for 40K iterations on the clean documents, and the detector network for 25K iterations (batch size 2048) on multi-defect broken layouts, up to three violations per each type. The correctors and router are trained independently, with no model's output being fed to another's input. More details in Appendix A.3.

Metrics: Existing layout metrics—LayoutFlow's and LACE's population-averaged Overlap/Alignment/FID, and LaySPA's unconditional per-layout reward—were built for generation and cannot tell whether one specific flagged element was fixed without disturbing another. We therefore introduce a gated evaluation suite: Satisfaction, Correction, and Preservation Rates computed over disjoint, non-overlapping element subsets (flagged, newly-flagged, and untouched), combined into a single Positional-Quality-gated score per defect type, plus Unbroken Drift—a do-no-harm metric that catches unflagged elements silently displaced into empty space. Full definitions and the disjoint-subset rationale are in Appendix A.4.

Configuration	Overlap				Alignment		
	Corr	PosQual	Pres	Gated Score	Corr	Pres	Gated Score
Act I — getting the corrector right, without touching the architecture							
Size-cond. generator (base)	20.0	4.7	94.9	48.0	3.4	68.2	34.1
+ refinement training (shared)	78.8	40.0	**98.8**	69.1	27.2	72.1	48.1
overlap specialist only	79.3	36.5	98.4	67.1	18.0	60.0	37.0
alignment specialist only	5.1	4.3	94.5	47.4	23.6	**82.1**	51.0
+ both specialists (sequential)	80.7	37.5	98.4	67.6	35.5	65.3	48.9
+ both specialists (sequential, multi-iterations)	**91.3**	33.5	98.5	65.9	45.9	59.7	51.6
Act II — turning two models into a pipeline							
+ reflection router	87.3	38.5	98.7	68.4	41.2	69.9	54.6
+ refinement mask	83.3	41.2	98.0	69.2	49.0	75.9	61.7
+ group-aware attention	87.0	46.1	98.6	72.1	**51.9**	76.5	63.3
+ edit-aware conditioning (full)	87.4	**48.2**	98.7	**73.2**	50.4	78.4	**63.6**

Table 1: Ablation from a generative model to the TidyUp pipeline on the internal validation split. Each row adds one component to the row above, except the two indented diagnostic rows. All values are percentages; higher is better, in **bold**, second best underlined. Positional Quality is defined for overlap only (Appendix A.4)

Method	Internal Dataset			Crello Dataset		
	Overlap Gated Score	Alignment Gated Score	UnbrokenDrift	Overlap Gated Score	Alignment Gated Score	UnbrokenDrift
LayoutFlow	48.00	34.10	**0.15**	46.26	39.07	**0.06**
LayoutFlow++	69.10	48.10	1.07	64.53	46.21	2.52
LaySPA	44.91	34.27	0.51	42.32	36.55	0.16
VASCAR	62.70	44.80	11.7	58.18	42.04	16.4
TidyUp	**72.10**	**63.30**	2.15	**66.93**	**55.51**	0.61

Table 2: Results on the internal and Crello datasets. Gated scores are higher-is-better, UnbrokenDrift lower.

5.2 Ablation Study

Table 1 traces nine configurations from a generative model to the full pipeline, showing that both gated scores must advance together. Act I shows why this is hard. Refinement training alone is enough to top the act on overlap (69.1), yet it trails the eventual router by 12% relatively on alignment, and the two specialist rows fail in mirror-image fashion—each strong on its own violation type, weak on the other—with a fixed sequential chaining of them inheriting both weaknesses rather than both strengths. Act II resolves this by making the assignment architectural, and the two violation types then improve through different channels: overlap through Positional Quality ($38.5 \rightarrow 48.2$, with Preservation already saturated near 98), alignment through Preservation ($69.9 \rightarrow 78.4$, Correction rising alongside rather than against it). It is the only stretch of the table where both scores rise monotonically, $(68.4, 54.6) \rightarrow (73.2, 63.6)$. Routing also beats iterated sequential chaining on overlap gated score (68.4 vs. 65.9) at roughly $4\times$ lower latency, since it invokes only the specialists a round needs; from Act II on we therefore cap the loop at three router rounds. Per-row analysis is in Appendix A.5.

5.3 Competitive Benchmarking

Table 2 compares TidyUp against three state-of-the-art layout and design generation systems—LayoutFlow (noise to layout) (Guerreiro et al., 2024), LaySPA (RL based LLM finetuned) (Li et al., 2026), and VASCAR (training-free iterative self corrector) (Zhang et al., 2024)—each repurposed for repair at inference following the recipe in the respective papers. LayoutFlow++ adds refinement-based training to LayoutFlow; VASCAR averages 15 GPT-5 (OpenAI, 2025) calls per design; the remaining systems are trained on the corresponding split via their public code. Since these splits carry no agentic state, all methods are evaluated in their *stateless* form. Across both datasets TidyUp attains the best gated score on both defect types: it corrects more while preserving more, despite carrying no vision head and running orders of magnitude fewer parameters than the Qwen- and GPT-scale baselines it outperforms. Figure 4 shows rendered outputs on final designs from multi-turn agentic editing sessions.

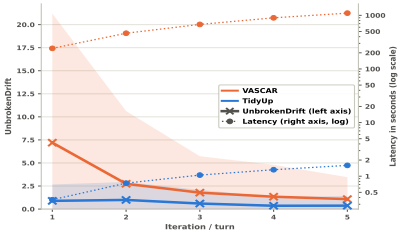

Figure 3: UnbrokenDrift and latency across correction rounds.

Cost of Iterative Correction. TidyUp and VASCAR are both *iterative*: each corrects, re-checks, and repeats until



Figure 4: Qualitative comparison on multi-turn agentic outputs. TidyUp only ever translates whole elements, so multi-part text is never re-ordered or re-spaced (the position-only invariant of §3)—clearest in the "Emily Branson" resume row, where the multi-line name and section blocks stay intact and correctly spaced, and again in the "PARTY" flyer row, while every other method jumbles the word blocks or breaks their spacing—VASCAR mangling them with and without state despite receiving the same state tokens. The bottom row traces one design through the iterative pipeline: each corrector output is followed by a fresh router pass, since one corrector's edits can stale the previous routing and must be re-assessed before the next corrector runs.

the loop terminates or a round budget is hit. We sweep a budget of 1–5 rounds and record UnbrokenDrift, latency, and cost. Lacking design foresight, VASCAR's reasoning agent rearranges layouts recklessly, incurring high and highly variable UnbrokenDrift even when averaged over five calls per round—at cumulative $1.2 per design and 1,000s over five rounds. TidyUp instead holds UnbrokenDrift low throughout, at negligible cost and under two seconds even at the full five-round budget—latency orders of magnitude lower. The deployed pipeline caps at three rounds (section 5.2), running in $\approx$ 1.2s per design.

6 Conclusion

We presented *TidyUp*, a tiny, vision-free pipeline that post-aligns any design agent's output—routing broken elements to two flow-matching correctors under correctness-preserving guards. By treating layout cleanup as *post-alignment* for a cheap specialist rather than the generalist that produced the design, TidyUp corrects more while preserving more than generative and self-correction baselines on both benchmarks, at sub-two-second latency and negligible cost. Its scope is deliberately geometric: never rendering the design, it leaves *content-aware* choices that genuinely need pixels—whether a caption occludes an image's salient region, and other creative decisions—to the editing agent. Closing the loop between the two—letting the agent call TidyUp as a fast inner geometric corrector mid-reasoning—is a natural next step.

REPRODUCIBILITY STATEMENT

We aim to make our results as reproducible as possible within the constraints of our data-access agreements.

Public data and defect-injection protocol: We report results over publicly available Crello dataset as a proxy for real-world design layouts (see Appendix A.1 for our justification for using Crello over PKU/RICO). We fully specify our synthetic defect-injection protocol—perturbation types, sampling ranges for training/validation/test, and the mover set unbroken set construction, in Appendix A.2, so that the training and evaluation distributions can be exactly reconstructed from the public dataset alone.

Code availability, pipeline training details: We are not releasing training or inference code at this time, as parts of our pipeline are subject to external disclosure restrictions. To compensate, we provide complete architectural and procedural detail in the Appendix A.3 — model architecture and layer-level configuration, full hyperparameter settings and training schedule and exact algorithmic descriptions of the flow-matching formulation and inference-time sampling procedure, sufficient for an independent implementation to reproduce our model from the public Crello data.

Metric definitions: All evaluation metrics (Satisfaction Rate, Correction Rate, Preservation Rate, Positional Quality, Gated Quality Scores, and UnbrokenDrift) are formally defined in Appendix A.4, including exact formulas and the constraint sets (Q_{sat}, Q_{corr} etc.) they operate over, so that results can be independently recomputed given model outputs.

ETHICS STATEMENT

Design automation: TidyUp performs a post-agentic edit cleanup of spatial violations in layouts (overlap, misalignment), rather than generating creative layouts from scratch. As with other design-automation tools, broader deployment could reduce demand for manual fine-tuning tasks currently performed by designers. We view TidyUp as a corrective/assistive layer intended to sit behind a human- or agent-driven design process, and does not intend to replace the creative decisions that precede it.

Bias in "good layout" priors: Our notion of a well-formed layout is learned from the reference distribution in training corpus (Crello/production system designs), and the corrected outputs $\mathcal{L}^*$ are only as unbiased as that corpus. Design conventions in Crello reflect the templates, categories, and stylistic norms of the platform it was sourced from, which may skew toward particular design traditions and under-represent others. However, since TidyUp does a narrower range of perturbations "assuming" it to be a repair-only task it's unlikely to penalize legitimate but less-represented design conventions as if they were violations. However we recommend that practitioners deploying TidyUp on out-of-distribution design cultures or scripts validate corrected outputs against locally appropriate design norms before production use, and consider fine-tuning / re-training on more representative corpora where feasible.

Data IP issues: All experiments use either the publicly available Crello dataset or production system designs which have been marked *safe* for training and are not platform's user-owned designs. TidyUp's vision-free formulation also limits the model's exposure to visual content, reducing the risk of consuming sensitive visual information (logos, photos, private text) that might appear in the original design assets during inference. We nonetheless, discard the design completely post its repair and the model does not log or audit any statistics about the document.

AI USE STATEMENT

In accordance with ICLR 2027 policy, we disclose the following uses of generative AI tools in this work.

Writing assistance: We used LLMs e.g., Claude to assist with editing prose for clarity and grammar, reformatting/typesetting LaTeX tables and figures. All AI-assisted text was was merely used as a reference by the authors, who take full responsibility for the accuracy and correctness of all content reported in the paper. No experimental results, numerical findings, or scientific claims were generated or fabricated by the LLM; the model was used purely as a writing and formatting aid

on author-provided content. No LLM was used to design experiments, analyze results, or draw scientific conclusions.

Code assistance: LLM tools were used to assist in writing auxiliary plotting/visualization scripts and boilerplate data-processing code. Core modeling contributions—the flow-matching formulation, model architecture, and training pipeline—were conceived, developed, designed and implemented by the authors.

REFERENCES

Jian Chen, Ruiyi Zhang, Yufan Zhou, and Changyou Chen. Towards aligned layout generation via diffusion model with aesthetic constraints. In *ICLR*, 2024.

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *UIST*, 2017.

Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer. In *ICLR*, 2026.

Julian Jorge Andrade Guerreiro, Naoto Inoue, Kento Masui, Mayu Otani, and Hideki Nakayama. Layoutflow: Flow matching for layout generation. In *ECCV*, 2024.

Hsiao Yuan Hsu, Xiangteng He, Yuxin Peng, Hao Kong, and Qing Zhang. Posterlayout: A new benchmark and approach for content-aware visual-textual presentation layout. In *CVPR*, 2023.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *ICLR*, 2024.

Shoma Iwai, Atsuki Osanai, Shunsuke Kitada, and Shinichiro Omachi. Layout-corrector: Alleviating layout sticking phenomenon in discrete diffusion model. In *ECCV*, 2024.

JooYoung Jang, Taegyeong Lee, Jihyeon Park, and Nojun Kwak. Ace: A self-correcting agentic canvas editor for multi-slide presentation automation. *EMNLP*, 2026.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *ICLR*, 2025.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. In *ICLR*, 2026.

Sha Li, Stefano Petrangeli, Yu Shen, and Xiang Chen. From pixels to policies: Reinforcing spatial reasoning in language models for content-aware layout design. In *ACL*, 2026.

Jieru Lin, Danqing Huang, Tiejun Zhao, Dechen Zhan, and Chin-Yew Lin. Spot the error: Non-autoregressive graphic layout generation with wireframe locator. In *AAAI*, 2024.

Tsiry Mayet, Pourya Shamsolmoali, Simon Bernard, Eric Granger, Romain Hérault, and Clément Chatelain. TD-Paint: Faster diffusion inpainting through time-aware pixel conditioning. In *ICLR*, 2025.

OpenAI. GPT-5.2. `https://openai.com/index/introducing-gpt-5-2/`, 2025.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 2024.

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational transformer: Toward zero-shot foundation models for relational data. In *ICLR*, 2026.

Evgenia Rusak, Patrik Reizinger, Attila Juhos, Oliver Bringmann, Roland S. Zimmermann, and Wieland Brendel. Infonce: Identifying the gap between theory and practice. In *AISTATS*, 2025.

Neta Shaul, Itai Gat, Marton Havasi, Daniel Severo, Anuroop Sriram, Peter Holderrieth, Brian Karrer, Yaron Lipman, and Ricky T. Q. Chen. Flow matching with general discrete paths: A kinetic-optimal perspective. In *ICLR*, 2025.

I-Chao Shen, Ariel Shamir, and Takeo Igarashi. Layoutrectifier: An optimization-based post-processing for graphic design layout generation. In *Computer Graphics Forum*, volume 44, 2025.

Zecheng Tang, Chenfei Wu, Juntao Li, and Nan Duan. LayoutNUWA: Revealing the hidden layout expertise of large language models. In *ICLR*, 2024.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Luran Wang, Chaoran Cheng, Yizhen Liao, Yanru Qu, and Ge Liu. Training free guided flow matching with optimal control. In *ICLR*, 2025.

Kota Yamaguchi. Canvasvae: Learning to generate vector graphic documents. In *ICCV*, 2021.

J Zhang, R Yoshihashi, S Kitada, A Osanai, and Y Nakashima. Vascar: Content-aware layout generation via visual-aware self-correction. *arXiv:2412.04237*, 2024.

Junyi Zhang, Jiaqi Guo, Shizhao Sun, Jian-Guang Lou, and Dongmei Zhang. Layoutdiffusion: Improving graphic layout generation by discrete diffusion probabilistic models. In *ICCV*, 2023.

Xu Zhong, Jianbin Tang, and Antonio Jimeno-Yepes. Publaynet: Largest dataset ever for document layout analysis. In *ICDAR*, 2019.

Zhiqiang Zhong, Kuangyu Zhou, and Davide Mottin. Harnessing large language models as post-hoc correctors. In *Findings of the ACL*, 2024.

Min Zhou, Chenchen Xu, Ye Ma, Tiezheng Ge, Yuning Jiang, and Weiwei Xu. Composition-aware graphic layout gan for visual-textual presentation designs. In *IJCAI*, 2022.

A Data

A.1 Datasets

We expand here on the dataset choice summarized in section 5.1.

Crello (Yamaguchi, 2021) is a corpus of vector graphic design templates harvested from an online design service. Each template is a canvas of typed elements—text, images, shapes, vector graphics, and video—whose schema exposes per-element opacity, a depth-ordered (z-indexed) stacking, and explicit element groups. Consequently overlaps in it are frequently *intentional* (a headline over a backing shape, a caption over a photo) rather than errors, which is exactly the distinction our legality rules make. It is the only public benchmark on which our defect synthesis and metrics are meaningful, and we use its standard released train/validation/test split throughout.

RICO (Deka et al., 2017) consists of mobile UI screens annotated with 25 widget categories. Each element carries only a type and a bounding box—no opacity, no z-order, no grouping—and placements are governed by a UI layout engine rather than by design intent, so it's devoid of creative overlaps.

PKU PosterLayout (Hsu et al., 2023) and **CGL** (Zhou et al., 2022) are content-aware poster benchmarks: a background image plus a handful of boxes drawn from three element types (logo, text, underlay). Like RICO they annotate type and box only, and their single design genre and small element counts do not reach the multi-element, multi-type arrangements we target.

PubLayNet (Zhong et al., 2019) annotates document pages with flat region boxes (text, title, list, table, figure). Regions are non-overlapping by construction and there is no z-order, so neither our overlap defects nor their legality rules transfer.

On any of these three, the group-rigid-body moves and depth-/opacity-aware violation filtering described below would be silently disabled, flattening the problem rather than testing it.

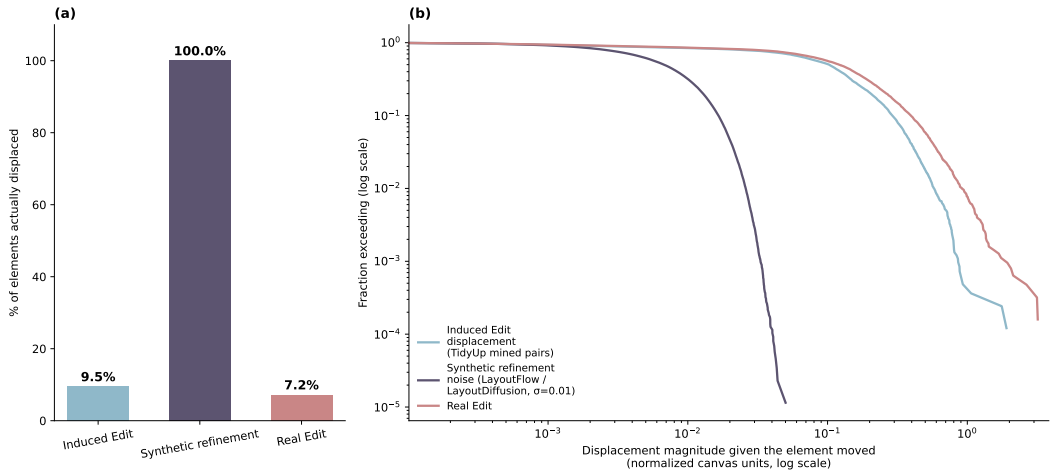

Figure 5: Corruption magnitude: induced edits vs. real agentic edits vs. synthetic noise. *Left*: the fraction of elements actually displaced by an *induced edit (TidyUp mined pairs)* (9.5%) and a *real multi-turn agentic edit* (7.2%), versus synthetic refinement noise, which perturbs every element (100%). *Right*: the distribution of displacement magnitude, given the element moved, for *induced edits* , *real agentic edits*, and synthetic *refinement noise (LayoutFlow / LayoutDiffusion,* σ=0.01*)* . Synthetic *refinement noise's* tail cuts off sharply near σ, while *real edits* exhibit non-trivial probability at much larger displacements, a pattern that *induced edits* effectively emulate.

Internal corpus. Our internal corpus of $\sim$0.5M clean design documents is both larger and structurally richer: designs average 20 elements and range from 4 to 55, and they exercise *user-grouped containers* at a scale no public benchmark reaches—the grouping primitive ubiquitous in PowerPoint, Keynote, Canva, Adobe Express, and Figma. Groups have no bounding box of their own and must translate rigidly as one body, which is what our group-aware attention and the leaf-based geometry tests as discussed in section 4.2 are designed for.

A.2 Defect Synthesis

Our correctors and router are trained entirely on *synthetic* defects, so the quality of that synthetic data decides what they learn. We therefore studied how defects actually arise in our production multi-turn editing agent, and compared them against the synthetic data pipelines used in prior layout work (Guerreiro et al., 2024; Zhang et al., 2023). Those pipelines add small Gaussian "refinement" noise to *every* element. Real agentic edits look nothing like this. As Figure 5 shows, real edits typically affect only a handful of elements, with movement magnitudes exhibiting a heavy-tailed distribution—whereas synthetic noise perturbs all elements by a small, symmetric amount. A model trained on uniform noise never sees the sparse, large-displacement defects it must actually repair.

This mismatch is why we do not reuse prior noise models. Instead, we designed a set of algorithms that *synthesize* breakages matching the real defect distribution, purely to build training data for the correctors and router. The pipeline runs offline, not at inference, and it does not generate new designs. It starts from a corpus of $\sim$0.5M clean designs and, using the procedures below, produces many single-defect (for correctors) and multi-defect, up to three (for router), broken variants of each one—in total $\sim$4M (broken, clean) pairs that the correctors learn to reverse and the router learns to classify.

The defects we synthesize are the ones most commonly observed in agentic layout outputs (Li et al., 2026; Zhang et al., 2024): element *overlap* and *misalignment*. These are also the dominant terms in the reward models used to train such agents; LaySPA (Li et al., 2026), for instance, scores a layout on inverse collision (overlap), alignment, spacing consistency, underlay/text-constraint, and IoU. Overlap defects primarily impair content visibility, while alignment defects compromise a design's creative focus and aesthetics. We treat these two as primary and cover the remaining two defect types as by-products: irregular spacing is handled by the overlap model, and off-canvas placement by the alignment model.

A.2.1 OVERLAP DEFECTS

We create overlap defects by exploiting the depth (stacking) order of elements: sliding one element until it covers part of another reproduces the occlusions seen in the agentic system.

Candidate pair enumeration. From a clean design, we enumerate every ordered sibling pair (mover, target)—both among top-level elements and among the members inside each user-group, with each user-group treated as its own container. We keep a pair only if three conditions hold: the resulting overlap would be a genuine defect rather than a valid design (see **Legality rules** below); the overlap is newly created, not already present in the clean design; and the mover has a clear straight-line path to the target, so it can slide into place without colliding with other elements on the way.

Legality rules. Whether an overlap is a defect is decided from element type and stacking order together. Translation never changes stacking order, so which element ends up occluded is fixed by the pair's relative z-order, regardless of which one moves. An image over text—or text slid beneath a higher image—hides the text, so we keep it as a defect. Text over an image is a common, intentional caption, so we treat it as valid and exclude it. Text–text, text–shape, and shape-over-text overlaps are always defects. User-groups need special handling because they have no bounding box of their own: we run all geometric tests on their leaf elements and translate the whole group rigidly as one body. Legality is checked for each leaf pair. Because the shift is rigid, we then re-validate the entire post-move geometry, keeping the move only if it produces at least one genuine (legal) overlap and creates no illegal collision.

Displacement and labeling. For an accepted pair, we slide the mover toward the target until it partially covers it. The shift is large enough to be clearly visible, but bounded so the result stays a *partial* overlap. We clamp the mover to its parent user-group box for within-group moves, and to the canvas otherwise, so nothing is pushed off-canvas. We apply this exhaustively, emitting one broken design per valid (mover, target) pair, so each clean design yields many single-defect variants.

A.2.2 MISALIGNMENT DEFECTS

We create alignment defects by first finding the intended alignment relationships between elements, then nudging one element off a relationship. We consider six alignment axes: left, horizontal-center, right, top, vertical-center, and bottom.

Aligned-set detection. From a clean design, we detect *aligned sets* on each of the six axes. An aligned set is a collection of elements that share an edge or center coordinate within a small pixel tolerance. To make sure we perturb only genuinely aligned, uncluttered sets, we additionally require the members of an aligned set to be mutually non-overlapping. We run this detection twice: among top-level elements, and, separately, among the members inside each user-group (treating each user-group as its own coordinate frame). User-group containers themselves can be potential members of any aligned set, and are translated rigidly as a single body.

Displacement and labeling. For each aligned set, we break the relationship by displacing one member off the shared line. We move it only along that alignment's own axis, by an importance-scaled amount that is floored so the shift stays clearly visible and genuinely clears the alignment tolerance. The displacement is contained within the enclosing user-group box for within-group breaks, and within the canvas otherwise. Unlike the overlap task, here we *reject* any move that would create a new illegal overlap (checked per leaf pair from element type and stacking order), so a misalignment defect never doubles as an overlap defect. We enumerate this exhaustively, emitting one broken design per displaced element while leaving at least one member of the set still aligned. Each clean design yields many single-defect variants, and each variant records the displaced element, the broken axis, and the applied offset.

The router is trained on a separate synthetic set in which a single document carries *both* overlap and alignment defects, so it learns to detect each defect and route it to the appropriate corrector.

Algorithm 1 Overlap-defect generation from one clean design.

Require: clean design D
Ensure: set $\mathcal{S}$ of single-defect broken designs
$\mathcal{C} \leftarrow$ CANDIDATES$(D, \text{top-level}) \cup$ CANDIDATES$(D, \text{within-group})$
$\mathcal{S} \leftarrow \emptyset$
for each candidate $(\textit{mover}, \textit{target}) \in \mathcal{C}$ **do**
  $D' \leftarrow$ SLIDEONTO$(D, \textit{mover}, \textit{target})$ ▷ one broken variant per accepted pair
  if $D' \neq \varnothing$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{D'\}$
  end if
end for
return $\mathcal{S}$

function CANDIDATES(D, scope)
  enumerate ordered sibling pairs $(\textit{mover}, \textit{target})$ in *scope*
  return pairs where sliding *mover* onto *target* is LEGAL, is a *new* overlap, and has a clear path
end function

function SLIDEONTO($D, \textit{mover}, \textit{target}$)
  slide *mover* toward *target* by a visible but bounded amount (partial overlap), clamped to its container or the canvas
  re-validate the full post-move geometry ▷ ≥ 1 genuine overlap, no illegal collision
  return the moved design, or $\varnothing$ if invalid
end function

function LEGAL(*mover*, *target*) ▷ by type and z-order
  image over text (incl. text beneath a higher image) $\rightarrow$ **defect**
  text over image (intentional caption) $\rightarrow$ **valid**, reject
  text–text, text–shape, shape-over-text $\rightarrow$ **defect**
end function

A.2.3 PSEUDOCODE

Algorithms 1 and 2 describe the overlap- and alignment-defect generation process for reproducibility; the terminology (units, leaf-based geometry, rigid user-group translation, and the type/z-order legality rules) is as defined there.

A.3 TRAINING

This appendix gives the full training schedules for the correctors and the router, together with the details omitted from the summary in section 5.1 to facilitate reproducibility.

Data splits. The $\sim$0.5M clean design documents are randomly partitioned into train/validation/test splits in a $0.98{:}0.01{:}0.01$ ratio. The synthetic broken variants of Appendix A are generated *per split*, so no clean document—and hence none of its broken derivatives—leaks across the partition. All broken pairs from the train split, train the models; those from the validation split produce the ablation study of Table 1; and those from the held-out test split are reserved for the competitor comparison of Table 2.

Corrector training. Each corrector is trained in the two stages of section 4.4: a base stage and a state-conditioned finetuning stage, each for 600K iterations at a batch size of 2048 and a learning rate of $5\text{e}{-4}$ on $2\times$H100 GPUs ($\sim$1 day per stage). Stage 2 differs from Stage 1 only in that the edit-aware state conditioning is enabled and its newly added state-processing modules use a learning rate of $5\text{e}{-3}$ ($10\times$ the rest of the network). Because the state module is zero-initialized (section 4.4), Stage 2 begins byte-for-byte identical to the Stage 1 corrector and only departs from it as finetuning proceeds; all remaining optimizer and model hyperparameters carry over from Stage 1 unchanged.

Router training. The router is likewise trained in two stages. First, its frozen base flow-matching generator (section 4.3) is trained for 40K iterations on the 0.5M clean documents. Second, the

Algorithm 2 Alignment-defect generation from one clean design.

Require: clean design D; axes $\mathcal{X} = \{$left, h-center, right, top, v-center, bottom$\}$
Ensure: set $\mathcal{S}$ of single-defect broken designs
$\mathcal{A} \leftarrow$ ALIGNEDSETS$(D,$ top-level$) \cup \bigcup_{\text{user-group } G}$ ALIGNEDSETS(G)
$\mathcal{S} \leftarrow \emptyset$
for each aligned set $A \in \mathcal{A}$ on axis a **do**
  for each victim $v \in A$ that leaves ≥ 1 member aligned **do**
    $\delta \leftarrow$ SHIFTOFFLINE(v, a)
    if $\delta \neq \varnothing$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{$APPLY$(D, v, \delta)\}$
    end if
  end for
end for
return $\mathcal{S}$

function ALIGNEDSETS(scope)
  for each axis $a \in \mathcal{X}$ **do**
    group elements sharing that edge/center coordinate within tolerance τ
    keep sets of ≥ 2 *mutually non-overlapping* members
  end for
  return the sets
end function

function SHIFTOFFLINE(v, a)
  shift v *only* along axis a by an importance-scaled, visibility-floored offset, clamped to its region
  return the offset, or $\varnothing$ if it would create a new illegal overlap ▷ inverse of the overlap guard
end function

detector network is trained for 25K iterations at a batch size of 2048 on a separate synthetic set in which each document carries *both* overlap and alignment defects, with up to three total violations of any type per document. This is the only set on which a single layout mixes defect types, so the router learns to detect and route each defect independently.

Independent training. The two correctors and the router are trained fully independently: there is no joint training, and no model's output is consumed as another model's input during training. Each component is optimized in isolation against its own synthetic data and objective, and the components are composed only at inference (section 4.6).

A.3.1 State Simulation and Augmentation

The three synthetic sets above supply geometry, but not the per-element state s_i that conditions our models on *what the agent changed* (section 4.2). We simulate that signal directly from the synthesis procedure: the elements the generator touched are exactly the changed set—the (mover, target) pair for an overlap defect and the (mover) for an alignment defect—so every broken variant comes with a ground-truth changed-element set for free, with the remaining elements marked unchanged. A real agentic turn, however, does not only move elements: it also adds new ones and deletes old ones between $\mathcal{L}^0$ and $\tilde{\mathcal{L}}$. To cover those transitions we dynamically flip the state of each element with probability 0.2 during training, which relabels untouched elements as newly added and changed elements as stale or removed. Because the flip is resampled every epoch, a single broken variant is seen under many different edit histories, and the models learn to rely on the geometry rather than to memorise a fixed state mask. To enable classifier-free guidance at inference, we additionally drop the state information for an entire document with probability 0.5 during training, so the same network learns both the state-conditioned and the unconditional velocity field and must fall back on geometry alone whenever the edit history is unavailable.

We further augment each pair by randomly removing elements from the input layout: with probability 0.5 we sample a drop fraction of up to 30% and delete that fraction of clean elements, applying the same deletion to both the clean and the broken member so their correspondence is preserved.

This yields partial design layouts—mid-composition documents in which only part of the final arrangement is present—which is the regime an editing agent operates in for most of a session. It also widens the element-count distribution of each corpus, exposing the correctors and the router to sparser contexts than the source designs alone provide.

A.4 METRIC DEFINITIONS

This appendix gives the full definitions of the evaluation metrics summarized in section 5.1, together with the worked failure cases that motivate each one and the rationale for computing certain metrics on disjoint subsets.

A.4.1 DESIGN ASSESSMENT PRELIMINARIES

Each evaluation example reuses the notation of section 3. The agent's clean prior layout $\mathcal{L}^0$—the agent's input—serves as the human-designed reference; the agent's candidate output $\tilde{\mathcal{L}}$, whose changed elements form the set $\mathcal{C}$, is the perturbed layout to be repaired; and $\mathcal{L}^*$ is the corrected layout produced by TidyUp or a competing model. All metrics are computed from the element geometry and relationships in these three layouts, and every model we compare is given the same $\tilde{\mathcal{L}}$ and the same flagged sets, so all methods are scored against an identical, model-independent definition of what was broken.

The changed set $\mathcal{C}$ of section 3 is obtained differently at evaluation time and at runtime. On the evaluation set it is exact: the defect synthesis of Appendix A records the elements the generator touched—the (mover, target) pair for an overlap defect and the displaced element for an alignment defect—so the flagged sets A_{ov} and A_{al} used below are ground truth, and no detection step enters any metric. On a runtime layout no such annotation exists, so $\mathcal{C}$ is instead *estimated* from the movers: an element is a mover if its position in $\tilde{\mathcal{L}}$ differs from its position in $\mathcal{L}^0$ by more than a fixed tolerance, large enough to absorb floating-point variation and small enough to catch a genuine displacement. Movers are elements the agent changed, not necessarily elements that are defective; it is the router (section 4.3) that flags and types the genuine violations among them, and because those predictions are proposals rather than guarantees the correctors are wrapped in the guards and do-no-harm bailout of section 4.6.

For overlap evaluation, both elements of a flagged (mover, target) pair belong to A_{ov}: although only one of them was displaced, the resulting collision involves both. For alignment evaluation, only the displaced element belongs to A_{al}; its stationary alignment partner is not included, since the relationship was broken by the displaced element alone. This distinction separates relationships affected by the perturbation from relationships that were already correct and should remain unchanged.

A.4.2 SOTA METRICS: WHY AREN'T THEY ENOUGH?

LACE Chen et al., 2024 and LayoutFlow Guerreiro et al., 2024 report Overlap, Alignment, and FID metrics over a population of generated layouts: FID measures how close the whole distribution of outputs is to a reference distribution, and their Overlap/Alignment are themselves *population-averaged* rates rather than per-example judgments. LaySPA Li et al., 2026 instead scores each layout individually, on inverse collision (overlap), alignment, spacing consistency, and underlay/text-constraint, but still unconditionally—its reward has no notion of a flagged vs. unflagged element, so it cannot tell whether a specific violation was fixed without disturbing something else. All three metrics fit generation-from-scratch, where there is no single ground-truth target and success means the aggregate output distribution looks plausible, or a layout scores well overall. They do not fit repair! TidyUp's task is defined per example and per element — did this flagged violation get fixed, and did this specific un-flagged element stay exactly where it was? Neither a distributional score nor an unconditional per-layout reward can answer either question — a model could report a perfect FID or LaySPA reward while every individual repair silently disturbs the wrong elements, precisely the dilution failure mode we will talk next and is built to avoid (a model that mishandles a few flagged elements but leaves the majority alone still looks good once averaged over everything).

Metric	Formula	Brief Description
Correction Rate	$\frac{1}{\lvert Q_{corr}\rvert}\sum_{\{i,j\}\in Q_{corr}} \mathbf{1}\left[\mathrm{box}_i(\mathcal{L}^*)\cap\mathrm{box}_j(\mathcal{L}^*)=\emptyset\right]$ where $Q_{\mathrm{corr}}=\{\{i,j\}\in N(\mathcal{L}^0):\mathrm{box}_i(\tilde{\mathcal{L}})\cap\mathrm{box}_j(\tilde{\mathcal{L}})\neq\varnothing\}$	Fraction of pairs that overlapped in $\tilde{\mathcal{L}}$ and are resolved in $\mathcal{L}^*$; measures whether the flagged collision itself was fixed.
Positional Quality	$\frac{1}{\lvert V_{ov}\rvert}\sum_{i\in V_{ov}}\mathrm{resolved}(i)\cdot\mathrm{IoU}(\mathrm{box}_i(\mathcal{L}^*),\mathrm{box}_i(\mathcal{L}^0))$ where $\mathrm{resolved}(i)=1$ only if element i satisfies all relevant non-overlap constraints in $\mathcal{L}^*$, else 0 and V_{ov} is the overlapping elements set	Gates on an element satisfying Correction conditions, then scores its corrected position against the reference; catches a "gamed" fix that clears every constraint but lands nowhere near the intended design.
Preservation Rate	$\frac{1}{\lvert Q_{pres}\rvert}\sum_{\{i,j\}\in Q_{pres}} \mathbf{1}\left[\mathrm{box}_i(\mathcal{L}^*)\cap\mathrm{box}_j(\mathcal{L}^*)=\emptyset\right]$ With $Q_{\mathrm{pres}}=\{\{i,j\}\in N(\mathcal{L}^0):j\notin A_{\mathrm{ov}},\ \mathrm{box}_i(\tilde{\mathcal{L}})\cap\mathrm{box}_j(\tilde{\mathcal{L}})=\varnothing\}$	Fraction of unflagged, previously non-overlapping pairs that remain non-overlapping; catches collateral damage to elements that were never part of the violation.
Gated Overlap Quality Score	$\frac{1}{2}(\mathrm{PositionalQuality}+\mathrm{PreservationRate})$	Combined overlap metric. Positional Quality already embeds Correction via its gate, so averaging it with Preservation Rate covers flagged-side accuracy and unflagged-side safety without double-counting.

Table 3: Overlap metrics: definitions used in evaluation

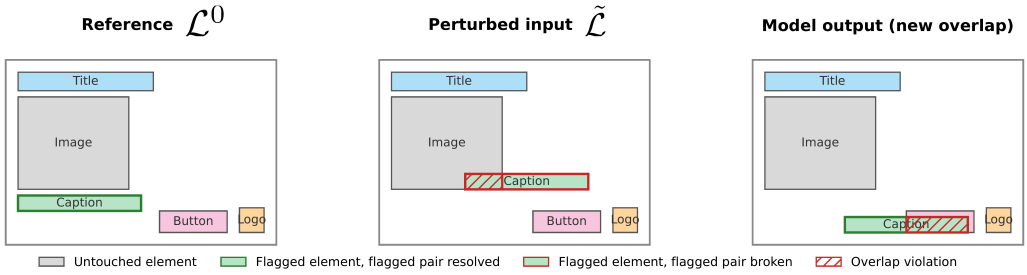

Figure 6: **Correction Rate only watches the flagged pair.** The Caption–Image collision flagged in *Perturbed input* is fully resolved in *Model output*, so Correction Rate reports 100%. But the same move plants Caption on top of Button, a pair Correction Rate was never computed over. Preservation Rate, which sums over all of Caption's reference non-overlap pairs, drops accordingly and catches the regression that Correction Rate cannot see.

A.4.3 Overlap metrics

Overlap evaluation must determine whether the corrected layout broadly satisfies its non-overlap constraints, whether the collisions present in $\tilde{\mathcal{L}}$ were removed, whether the affected elements returned to appropriate positions, and whether the correction damaged unmodified parts of the layout. We therefore report the following complementary metrics as shown in Table 3.

Gated Overlap Quality Score. Positional Quality already embeds Correction Rate (via its $\mathrm{resolved}(i)$ gate) and adds the ground-truth positional comparison that neither can make; Preservation Rate supplies the one signal Positional Quality structurally cannot. Averaging these two disjoint-subset scores—rather than pooling all pairs into one metric, for the reason given above—produces a single number that reflects both the quality of the intended correction and the absence of collateral damage elsewhere in the layout.The combined overlap metric is

$$\mathrm{GatedOverlapQualityScore}=\frac{1}{2}\left(\mathrm{PositionalQuality}+\mathrm{PreservationRate}\right).$$

A.4.4 Alignment metrics

Alignment violations are also relational, but the relevant constraint is different. Instead of requiring two elements not to intersect, an alignment relationship requires corresponding edges or centers to coincide within a fixed tolerance. Unlike overlap evaluation, alignment metrics are computed

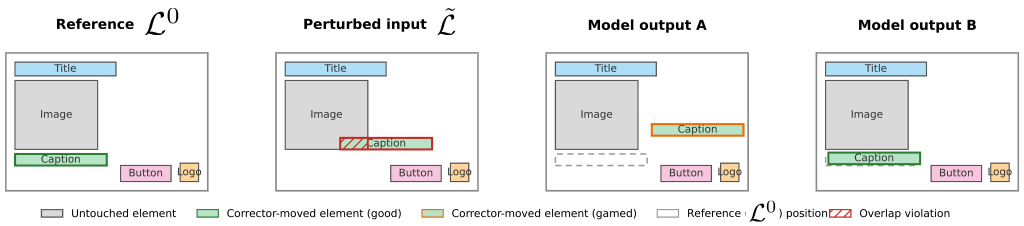

Figure 7: **Preservation and Correction Rate cannot tell these two corrections apart.** Starting from the same flagged Caption–Image collision, *Model output A* drops Caption into an empty region unrelated to anything it was captioning, while *Model output B* returns Caption close to its reference position. Both are non-overlapping everywhere, so both score 100% on Preservation Rate and 100% on Correction Rate. Only Positional Quality, which compares against the ground-truth position $\mathcal{L}^0$, tells them apart.

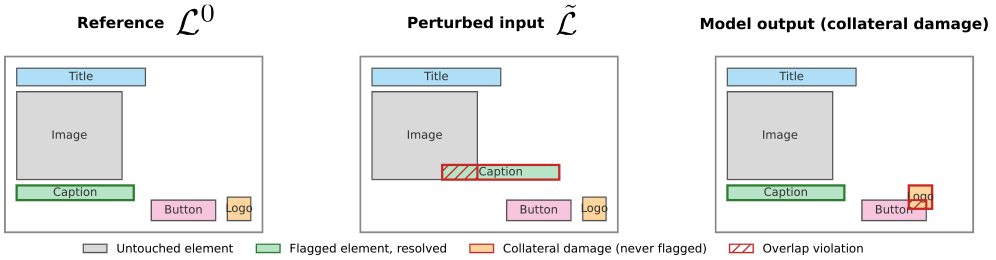

Figure 8: **A perfect fix can still come with damage nobody was watching for.** Caption–Image, the flagged pair, is resolved exactly: Correction Rate, and Positional Quality all score this element perfectly. But Logo—never part of any flagged violation, never touched by A_{ov}—is shifted into Button, a pair whose Positional Quality was never computed for. Only Preservation Rate, which is defined precisely over unflagged pairs, catches it.

only over pairs that are aligned in the reference layout, rather than over all pairs in the document. Complete list of metrics used to measure alignment fixes are discussed in Table 4

Why Alignment Does Not Use Positional Quality. Overlap resolution admits many geometrically valid but semantically incorrect solutions: any sufficiently distant empty location can remove a collision. Alignment is more constrained. In our perturbations, the alignment partner generally remains stationary, and restoring the reference alignment anchors the displaced element relative to that partner. Consequently, Correction Rate provide stronger information about positional recovery than their overlap counterparts. A separate alignment version of Positional Quality therefore provides substantially less additional information in this setting (Figure 10).

Gated Alignment Quality Score. The **Gated Alignment Quality Score** is the combined alignment metric: the unweighted arithmetic mean of Correction Rate, and Preservation Rate, computed per evaluation example and then averaged across the dataset,

$$\text{GatedAlignmentQualityScore} = \frac{1}{2}\left(\text{PositionalQuality} + \text{PreservationRate}\right).$$

All metrics are reported as percentages, and higher values indicate better performance.

A.4.5 Do NO-HARM metric - Unbroken Drift:

Preservation Rate, for both overlap and alignment, is a relational metric: it is only ever computed over a pair or relationship, and only ever detects damage that manifests as a new violation of that pair or relationship. This leaves a gap. A model can move an unbroken element into genuinely empty space – touching nothing, breaking no alignment – and both Preservation Rates score it perfectly, even though the element was never supposed to move at all (Figure 12). Unbroken Drift closes this

Metric	Formula	Brief Description
Correction Rate	$\frac{1}{\lvert Q_{corr}\rvert}\sum_{(i,j,a)\in Q_{corr}} \mathbf{1}\big[(i,j,a) \text{ holds in } \mathcal{L}^*\big]$ where $Q_{corr} = \{(i,j,a) \in G(\mathcal{L}^0) : (i,j,a) \text{ does not hold in } \tilde{\mathcal{L}}\}$	Fraction of alignment relations broken by the perturbation that are restored in the corrected layout.
Preservation Rate	$\frac{1}{\lvert Q_{pres}\rvert}\sum_{(i,j,a)\in Q_{pres}} \mathbf{1}\big[(i,j,a) \text{ holds in } \mathcal{L}^*\big]$ where $Q_{pres} = \{(i,j,a) \in G(\mathcal{L}^0) : (i,j,a) \text{ holds in } \tilde{\mathcal{L}},\ j \notin A_{al}\}$	Fraction of untouched alignment relations that remain intact after correction (no-harm check).
Gated Alignment Quality Score	$\text{GAQS} = \text{PresRate} \cdot (\lambda \cdot \text{SatRate} + (1-\lambda) \cdot \text{CorrRate})$ gates the weighted Sat/Corr blend by Preservation Rate so gains cannot come at the cost of new violations	Single scalar summarizing alignment quality, penalized whenever preservation is imperfect.

Table 4: Alignment metric definitions used in evaluation

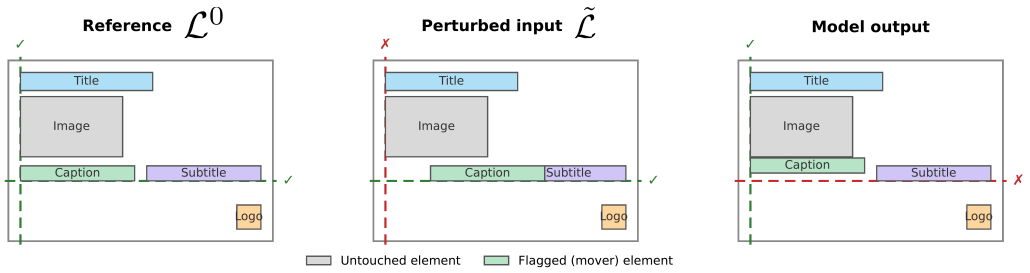

Figure 9: **Correction Rate only watches the flagged relationship.** The caption has two reference relationships: a left-edge alignment with the image and a baseline alignment with the subtitle. The perturbation breaks only the first. The model restores the image alignment—Correction Rate scores 100% for the one relationship it was measured against—but, in doing so, shifts the caption enough to break the subtitle baseline alignment, which had never been flagged and which Correction Rate never checks. Preservation Rate, which considers both relationships on the affected element, drops to 50% and catches the regression.

gap by scoring position directly, the same way Positional Quality complements Preservation and Correction Rate on the flagged side of the evaluation, but applied here to the un-flagged side.

Let $M = \{i : \|\text{box}_i(\tilde{\mathcal{L}}) - \text{box}_i(\mathcal{L}^0)\| > \tau\}$ be the movers and $U = \{1,\dots,N\} \setminus M$ the unbroken elements – those the perturbation never touched. Since $\text{box}_i(\tilde{\mathcal{L}}) \approx \text{box}_i(\mathcal{L}^0)$ for every $i \in U$ by definition, either layout serves equivalently as the reference position. With $\text{pos}_i(\mathcal{L})$ denoting element i's center coordinate in layout $\mathcal{L}$ (normalized to the unit canvas),

$$\text{UnbrokenDrift} = \frac{1}{|U|}\sum_{i\in U} \|\text{pos}_i(\mathcal{L}_o) - \text{pos}_i(\mathcal{L}_c)\|_2.$$

Unlike every other metric in this appendix, lower is better: UnbrokenDrift is a distance, so a value of zero means every unbroken element landed exactly where it started.

A.5 Ablation Analysis

We expand here on the ablation summarised in section 5.2, walking through Table 1 row by row.

Act I. The size-conditioned generator's overlap Preservation of 94.9 is an artifact rather than a merit. Injecting the broken layout at t_{low} leaves the learned velocity field near zero, so every element—broken or intact—moves only slightly; the model preserves the correct elements because it does not meaningfully move anything at all, which is equally why it repairs nothing (20.0 Correction, 4.7 Positional Quality). Preservation is therefore satisfiable by inaction, and only reading it against Correction exposes the difference. After refinement training the Preservation of 98.8 is earned while genuinely relocating elements. The two diagnostic rows then fail in complementary ways: the overlap specialist reaches 67.1 on overlap but 37.0 on alignment, while the alignment specialist

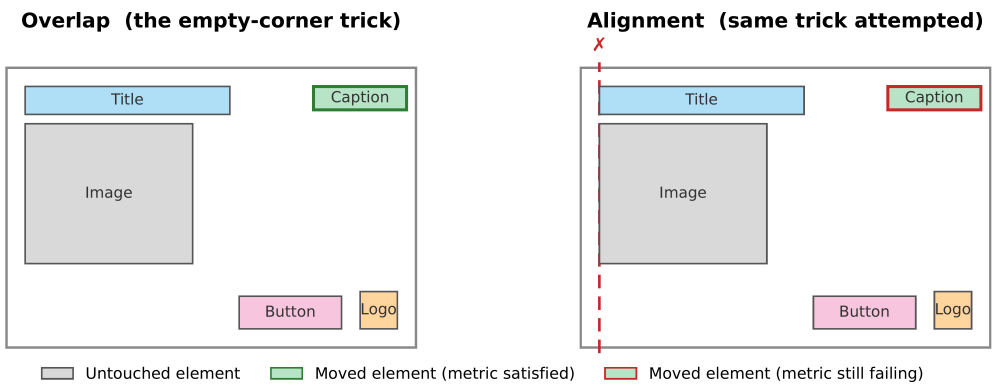

Figure 10: **Alignment is harder to game than overlap.** The open-corner trick from Figure 7 trivially satisfies non-overlap: any sufficiently empty region works, so Correction Rate reads 100% regardless of where the element lands. The same corner does nothing for alignment, which requires one specific coordinate—here, the image's left edge—to be recovered; alignment Correction Rate reads 0% until that coordinate is actually matched. This is why alignment restoration is a stronger positional signal on its own and does not need a separate Positional-Quality-style metric.

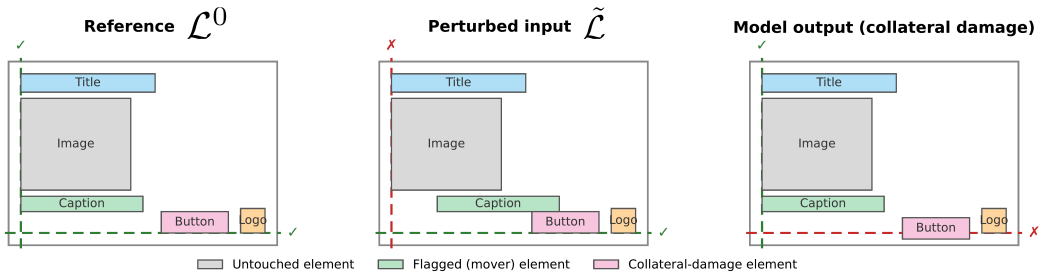

Figure 11: **Restoring a flagged alignment can break a different, unflagged pair.** The caption's left-edge alignment with the image is broken and then correctly restored, so Correction Rate reads 100% since it is never computed over the button or the logo. But the fix nudges the button off the logo's baseline—a bottom-alignment relationship between two elements that had nothing to do with the caption and was never flagged. Only Preservation Rate, defined precisely over unflagged relationships, exposes this collateral damage.

posts the best alignment Preservation of Act I (82.1) yet falls to 47.4 on overlap—no better than the untrained base. Chaining them inherits neither strength cleanly: alignment Preservation drops from 82.1 to 65.3, because the second pass spends its capacity displacing what the first had settled.

Act II. The two violation types improve through different channels. Overlap Preservation is already at its ceiling (98.0–98.7) and stays there, so the gains are carried almost entirely by Positional Quality, which climbs monotonically $38.5 \rightarrow 41.2 \rightarrow 46.1 \rightarrow 48.2$: each added signal tells the corrector more about the document's intended structure, and that shows up as more accurate placement rather than more corrections. Alignment improves through the opposite channel—every preservation-oriented mechanism raises Preservation in turn ($69.9 \rightarrow 75.9 \rightarrow 76.5 \rightarrow 78.4$), with Correction rising alongside it ($41.2 \rightarrow 49.0 \rightarrow 51.9$) rather than at its expense, since knowing which elements to hold fixed is what leaves the remaining relationships recoverable. The final step gives up 1.5 points of alignment Correction to gain 1.9 of Preservation, a net gain only because the gated score refuses to reward corrections bought with collateral damage.

Iteration budget and its cost. Running each specialist sequentially for several passes does lift some metrics, but pays for it twice—in quality and in latency. Iterating the sequential schedule pushes overlap Correction to its column maximum (91.3), yet the gated score *falls* to 65.9, the lowest of any repaired row, because the extra passes resolve collisions by displacement that Positional Quality

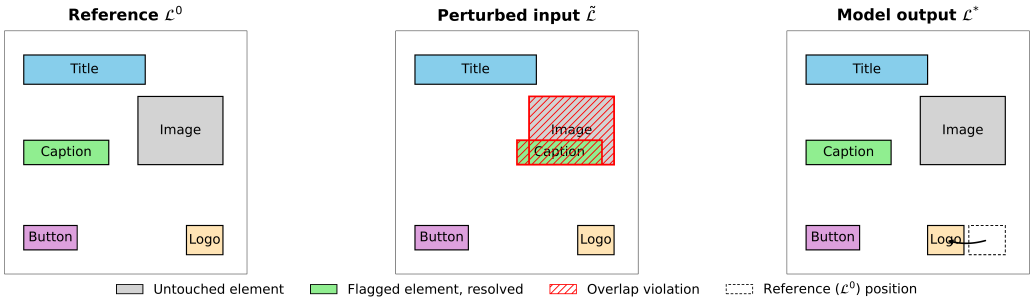

Figure 12: **Why Unbroken Drift is needed: Left ($\mathcal{L}^0$):** reference layout. **Center ($\tilde{\mathcal{L}}$):** perturbed input, with the Caption–Image overlap flagged (red hatching); the Logo is untouched by the perturbation and is not a mover. **Right ($\mathcal{L}^*$):** model output. The flagged overlap is resolved (green), so every gated overlap metric in Appendix A.4.3 – Correction Rate, Positional Quality, Preservation Rate – scores this correction perfectly. But the Logo was never in A_{ov} to begin with, so its displacement (black arrow, from its $\mathcal{L}^0$ position, dashed outline) falls outside every existing metricś purview, Unbroken drift registers the displacement as nonzero error.

(33.5) and Preservation both penalize: correction bought with drift. The latency cost compounds this, since the naive schedule invokes both correctors on every round regardless of what is broken, so a document with a single defect type still pays the full cost of the other. The router avoids both failure modes—it dispatches only the specialists a round needs, reaching a higher overlap gated score (68.4 vs. 65.9) at roughly $4\times$ lower latency. These gains saturate quickly—the third round rarely moves the gated score—so from Act II onward we cap the loop at three router rounds, where the quality-latency trade-off flattens.

Why both columns must move. The shared refinement model makes the trap concrete: its overlap gated score of 69.1 is the highest in Act I and edges out even the routed pipeline's 68.4, yet its alignment score of 48.1 sits 12% relatively below the router's 54.6. Read one column at a time, the router looks like a regression; read jointly, it is the first configuration that stops trading one violation type against the other. The alignment specialist falls into the same trap from the other side. Act II is the only stretch of the table where both scores advance together, from (68.4, 54.6) to (73.2, 63.6).

A.6 Tidy Up Architecture and Methodology Details

A.6.1 Correctors: Why repair vs regeneration interleaving prevents mode collapse:

At inference we place the broken layout at t_{low} and integrate the field to $t{=}1$ with a few Euler steps. However during the training, leveraging *only* tge repair samples, however, would collapse the model: under the linear path the regression target is $x_1 - x_0$, and in a broken variant most elements are already clean ($x_0{\approx}x_1$), so their target velocity is ≈ 0. A field regressed mostly onto such near-zero targets learns to predict near-zero velocity almost everywhere, shrinking its output range until it can no longer transport noise into a design—forgetting the design prior. Interleaving the generation samples, whose targets are large and structured, prevents this collapse, while the repair samples teach the field to invert agent errors.

This bounded-interval repair admits an optimal-control view of the pretrained flow (Wang et al., 2025). Starting the uncontrolled field from $\tilde{x}$ at t_{low} balances defect removal and input preservation: smaller values allow greater change, while larger values better preserve the input.

A.6.2 Correctors: Group-aware attention

User-group is a coherent group of elements that is explilictly marked by a user to enhance aritistic value / branding guidline / structural information flow of a design. It is very important to maintain this coherence, while ensuring our method can correctly interpret and leverage this information. For a non-user-group element, any user-group in the design should be interpreted as an atomic unit, meaning individual elements within that user-group should carry little weight relative to it. Similarly

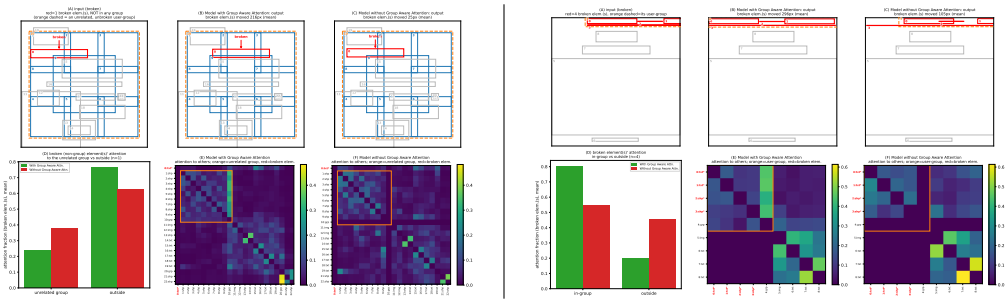

Figure 13: **Significance of Group Aware Attention: Orange Box** marks the user-group. **Left:** In the Group-aware attention model, a mis-aligned non-user group element (*indexed* **0**) attends more to non-user group elements **(11 - 22)** than to other unrelated user-group elements **(1 - 9)**. **Right:** In contrast, if user-group elements **(0 - 4)** are mis-aligned, they attend more within a user-group than to other non-user group elements **(5 - 8)**. In either case, Group-aware attention facilitates the model in understanding user-group structure and surfaces the right signals for correcting the design.

elements within a user-group should primarily interact with each other rather than unrelated non-user-group elements. To encode this understanding into our solution, we introduce group-aware attention mechanism. Figure 13 visualizes how the group-aware model interprets the user-group and leverage this information, compared to a model trained without group-aware attention. We considered alignment corrector model for this analysis. In the first scenario, a mis-aligned non-user group element attends more to other non-user group elements than to unrelated user-group elements in the group-aware model, resulting in better correction compared to the model without group-aware attention. In the second scenario, mis-aligned user-group elements attends more within its group than to other non-user group elements, again leading to better correction compared to the non-group-aware model.

A.6.3 Router: Architecture and Training

Architecture: For training the router we tokenize each element as a short sequence of discrete tokens. These are its category, its subtype, and its box quantized into a shared vocabulary of 512 bins for x, y, w, h, together with its stacking order. The backbone otherwise follows the token-reinitialization module of Layout-Corrector (Iwai et al., 2024)—a four-layer, eight-head transformer whose latent dimension shrinks with every layer—aggregating each element's attribute tokens into a single per-element representation h_i.

Router Training: Unlike the correctors, which act in continuous coordinate space, the router works over the *quantized* token representation —a deliberate choice, not merely an inheritance from Layout-Corrector. In continuous space, defect detection is hypersensitive: an arbitrarily small displacement can flip a clean/broken decision, making the detector noisy and hard to train. Binning collapses sub-bin jitter, so only a meaningful displacement changes an element's tokens; a continuous-space router did not match this discrete one in our experiments.

Router Training for high recall. Router's element labels are dominated by NONE as most being clean. The imbalance is countered by: (1) use of *focal* cross-entropy (γ=2) to down-weigh easy samples; (2) explicitly down-weigh the NONE class and (3) apply per-batch inverse-frequency class weights. An auxiliary supervised InfoNCE term (Rusak et al., 2025) on the per-element embeddings pulls together elements of the same break type and pushes apart different types further separating overlap issues from alignment in representation space. Router is tuned towards high *recall* or over-flagging: as the defect it misses will be ignored by the correctors. A false positive, by contrast, is comparatively harmless — a correctly-placed element already lies near the design manifold, so the repair flow barely moves it. The router has $\sim$14M trainable parameters.

Routers Conditioning Correctors with Masks Apart from selecting which corrector model to run, the sets $\mathcal{O}$ and $\mathcal{A}$ also seed the corresponding corrector as per-element mask m. A corrector's learned prior is only trustworthy near the region of the design manifold its training data covers. A design with several simultaneous violations sits off the manifold along many coordinates at once, and projecting all of them back in a single pass drives the flow through low-density regions where the learned

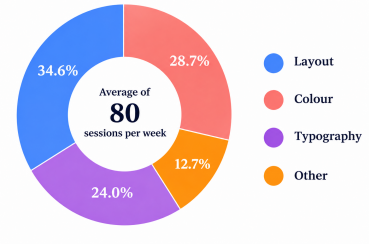

Figure 14: Distribution of issues with designs an agentic multi-turn edit workflows after 4 or more editing turns. Among the other issues, layout issues are the most prevalent.

velocity is poorly constrained, so the trajectory overshoots or settles on an implausible configuration. The mask m enforces exactly this restriction: it confines a pass to a non-overlapping subset of violating elements, keeping every correction a short, well-supported step along the manifold, while the iterative router loop supplies these subsets one at a time.

A.6.4 A DO-NO-HARM INFERENCE CALCULUS

Around this core, a set of correctness-preserving guards handles cases that a purely geometric repair can mishandle—for instance, leaving rotated elements and exact duplicates undisturbed, moving a broken group as a rigid body, and pulling text that an edit pushed off-canvas back inside. The guards are conservative by construction, only ever preventing or undoing harm, and preserve the position-only invariant of section 3: every stage merely translates elements. Bailout guardrails can be updated based on the respective agentic issue distribution/prior and the design data distribution.

A.7 AGENTIC MULTI-TURN EDIT STATISTICS

In our analysis of 724 editing sessions on a *production system* over a period of 9 weeks, 34.6% of issues were due to at least one overlap or alignment violation by the 4th editing turn of the agent. Figure 14 illustrates the distribution of errors/issues in the multi-turn agentic editing sessions.