ANCHORFLOW: LEARNING ANCHOR PLACEMENT FOR FAITHFUL AND EDITABLE SVG RECONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Raster-to-SVG reconstruction requires faithful geometry and a compact control structure for editing. A central challenge is deciding where to place anchors: raster appearance alone does not determine how a contour should be divided into Bézier segments. We present AnchorFlow, which learns anchor placement from designer-authored SVGs to reconstruct accurate curves with sparse controls. Our key idea is a sparse anchor field that jointly encodes contour geometry and reference segment junctions, including those along smooth contours. An anchor decoder predicts explicit anchor proposals from features learned under field supervision. These proposals guide boundary-constrained fitting and local refinement to recover cubic Bézier paths. On clean single-path inputs, AnchorFlow achieves 99.52% mean IoU while using 56.6% fewer anchors on average than AdaVec, with lower boundary error and closer agreement with source-SVG anchor layouts. Under boundary perturbations, it maintains high fidelity with limited anchor growth. Integrated into a component-based pipeline, the same path module also produces compact, faithful full-image reconstructions. On four local-editing tasks, our outputs require less median active time and fewer actions than AdaVec and LIVE while retaining high target-shape accuracy.

1 INTRODUCTION

Scalable Vector Graphics (SVG) preserve geometric structure, allowing artwork to be scaled and edited through its shapes and control points. Image vectorization reconstructs this representation from raster images, making existing artwork available for further editing. In icons and illustrations, closed paths often describe the outlines of individual components. Reconstructing such images involves organizing components and their layer order, as well as recovering the curve structure of each component. Even when component boundaries are available, recovering accurate paths with compact, editable controls remains a distinct challenge. We address this path-level reconstruction problem.

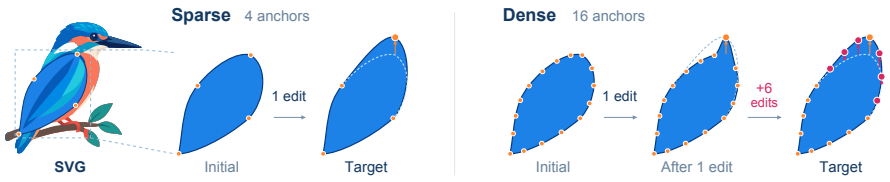

Figure 1: **Anchor layout affects editing.** The same initial wing outline is represented with 4 or 16 anchors. The illustrated adjustment involves one anchor group in the sparse representation and seven groups $(1 + 6)$ in the dense representation.

Visually similar paths can nevertheless have very different control structures. Anchors, the junctions between consecutive curve segments, determine how these segments are distributed along an outline. Figure 1 illustrates how this affects editing. Both paths reproduce the same initial wing outline, but moving one anchor group reshapes a broad portion of the sparse contour, whereas the illustrated adjustment involves seven groups in the dense path. Coordinating more controls can make such adjustments laborious. Simply removing anchors, however, can lose corners, narrow features, or

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.

changes in curvature. How can we reconstruct a contour accurately while keeping its controls compact and manageable?

We approach this question through the segmentation of the observed contour. For a closed Bézier path, consecutive anchors specify the endpoints of each segment. Selecting and ordering an initial anchor set along the contour therefore determines both the number of segments and the portion of the contour assigned to each one. Once this segmentation is specified, each curve can be fitted to the corresponding portion of the input image boundary. Our intuition is to predict these segment junctions first, then reconstruct the curves between them using the observed boundary shape.

The same contour admits multiple segmentations, so visual agreement alone does not uniquely determine these junctions. Designer-authored SVGs provide examples of how shapes are divided into curve segments, including junctions on smooth arcs without visible corners. We introduce ANCHORFLOW to learn a contour-conditioned anchor-layout prior from these examples. Its key representation is a *sparse anchor field* that combines contour support with localized peaks at reference anchor positions. An anchor decoder reads features learned under field supervision to predict explicit anchor proposals. These proposals guide contour segmentation, while the observed boundary constrains curve fitting and refinement. This combination of learned segmentation and geometric fitting forms a reusable path reconstruction module, which we apply to the extracted components in our full-image pipeline.

We evaluate reconstruction quality, control structure, and editing performance. On 2,400 clean paths, ANCHORFLOW achieves 99.52% mean IoU with 9.53 anchors on average, using 56.6% fewer anchors than AdaVec (Zhao et al., 2025) and agreeing more closely with source-SVG anchor layouts. It maintains high fidelity with limited anchor growth under boundary perturbations and produces compact full-image reconstructions. Our ablation studies further show that learning where to place anchors improves reconstruction accuracy while keeping the resulting paths compact. In an 18-participant study of four selected local-editing tasks, our outputs yield lower median active time and fewer actions than AdaVec and LIVE (Ma et al., 2022), while retaining high target-shape accuracy. Our contributions are:

- **Learning anchor placement through a sparse field.** We introduce a spatial representation that jointly encodes contour geometry and reference segment junctions, enabling anchor-layout learning from designer-authored SVGs, including junctions along smooth arcs.
- **Anchor-guided SVG reconstruction.** We combine learned anchor proposals with boundary-constrained curve fitting and bounded refinement to reconstruct accurate, compact Bézier paths, and extend this approach to full-image SVG reconstruction.
- **Evaluation of reconstruction and editing.** We demonstrate faithful, compact reconstruction on clean and perturbed paths and full images, supported by controlled placement comparisons and an editing study on four selected tasks.

2 RELATED WORK

Image vectorization reconstructs the shapes in a raster image as vector paths. Existing methods approach this task through boundary fitting and raster optimization, component decomposition and adaptive simplification, or learned vector representations.

Boundary fitting and raster optimization. Tracing methods such as Potrace, AutoTrace, and VTracer extract bitmap boundaries and convert them into vector paths through curve approximation and simplification (Selinger, 2003; AutoTrace Project, 2024; Vision Cortex, 2024). They construct curves using local boundary geometry and approximation error. Optimization-based methods render a candidate SVG and adjust its parameters to reduce disagreement with the input image. DiffVG provides differentiable rasterization for this process (Li et al., 2020); LIVE progressively adds and optimizes paths (Ma et al., 2022), Optimize & Reduce removes less important shapes during optimization (Hirschorn et al., 2024), and Vector Scaffolding organizes optimization across scales (Lee et al., 2026). These methods provide direct geometric or image-based constraints on shape reconstruction. Control structures depend on path initialization, curve budgets, and simplification criteria: similar reconstructed appearances do not imply the same curve segmentation or anchor layout.

Component decomposition and adaptive simplification. Full-image vectorization must also determine which regions should be represented as separate components or paths. SuperSVG groups neighboring pixels with similar appearance into superpixel regions and reconstructs vector graphics within them (Hu et al., 2024); SAMVG and SGLIVE use image segmentation to guide path construction (Zhu et al., 2023; Zhou et al., 2024). These approaches provide component-level organization for complex images. AdaVec further combines component decomposition with adaptive control-point simplification: it traces component contours, merges neighboring curve segments using geometric criteria, and performs subsequent optimization (Zhao et al., 2025). Adaptive vectorization therefore already addresses redundancy in both paths and control points. Our full-image pipeline uses an AdaVec-based component decomposition frontend and applies our path reconstruction module to each extracted component. Within each component, we investigate how segmentation regularities learned from designer-authored SVGs can provide an anchor-layout initialization for subsequent geometric fitting and refinement.

Learned vector representations and reconstruction. Learning from vector examples provides data-driven priors for path organization and control-point placement. SVG-VAE and DeepSVG learn structural regularities by modeling drawing commands and their organization (Lopes et al., 2019; Carlier et al., 2020); StarVector, OmniSVG, and InternSVG further use multimodal models to generate SVG code or command sequences from input images (Rodriguez et al., 2025; Yang et al., 2025; Wang et al., 2026). These advances show that designer-authored SVGs can provide structural priors beyond local boundary geometry. However, direct SVG generation predicts both path structure and curve coordinates, and command-sequence prediction does not itself explicitly constrain individual curves to follow the observed boundary. Consequently, a result with coherent structure and similar overall appearance can still exhibit local contour shifts or distorted details. ANCHORFLOW learns an anchor-layout prior from reference SVGs and combines it with boundary-constrained curve fitting and refinement, allowing learned segmentation patterns to guide the control structure while the observed contour constrains the geometry.

3 METHOD

ANCHORFLOW reconstructs a compact, closed cubic Bézier path from a rasterized component contour. Field supervision learns contour and anchor-layout features, an anchor decoder predicts segment junctions, and boundary-constrained fitting and refinement recover the curves between them. Figure 2 summarizes the pipeline.

3.1 ANCHOR-FIELD LEARNING

Representing shape and segment junctions. We rasterize the observed boundary into a 512×512 contour image X as input to AFNet. From the corresponding clean source contour Γ and its source-SVG anchor set A^*, we construct a target field F^* that combines a low-amplitude contour band c_Γ with Gaussian anchor peaks:

$$F^*(u) = \max\left\{c_\Gamma(u), \max_{a \in A^*} \exp\left(-\frac{\|u-a\|_2^2}{2\sigma_a^2}\right)\right\}. \tag{1}$$

Here u is a pixel coordinate, a is a reference anchor, and σ_a controls the anchor-peak width. The contour band supplies boundary context, while the peaks mark the reference segment junctions.

Field supervision. AFNet is a convolutional encoder–decoder with skip connections that preserve spatial detail. With network parameters θ, it predicts $F_\theta(X)$ and is trained using

$$\mathcal{L}_{\text{field}} = \frac{1}{HW}\sum_u \left(5e_u^2 + |e_u|\right), \qquad e_u = F_\theta(X)_u - F^*(u), \tag{2}$$

where H and W are the field height and width, and e_u is the prediction error at pixel u. For perturbed inputs, target fields are constructed from the corresponding clean source contours and anchors using the same geometric transform as the input.

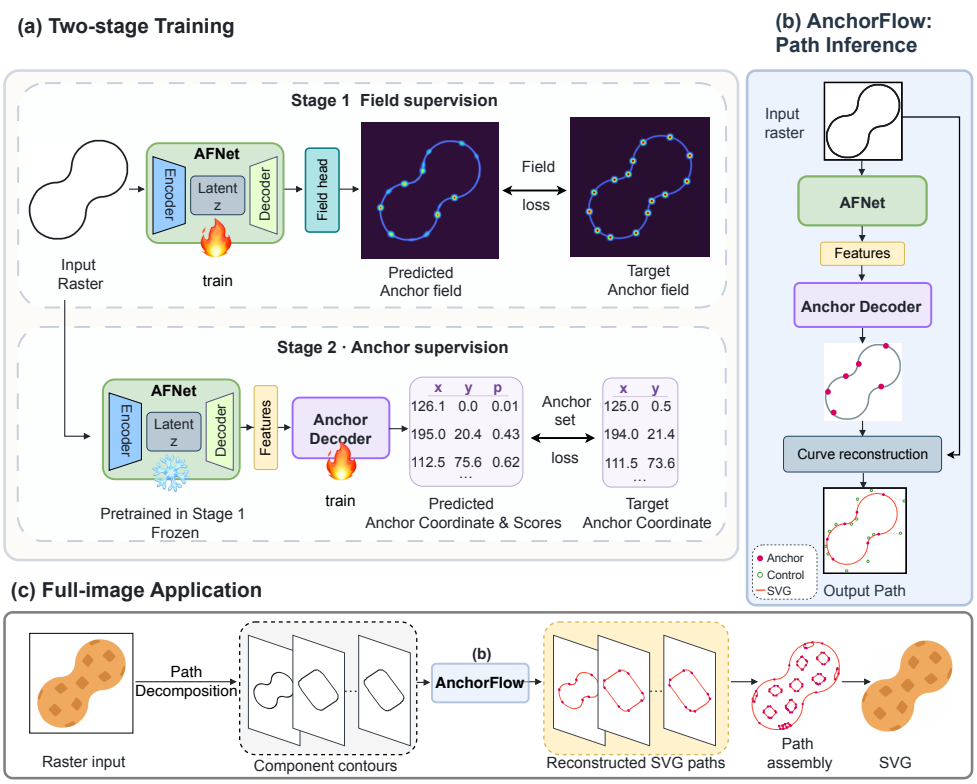

Figure 2: **Overview.** (a) Field supervision trains AFNet to encode contour geometry and reference anchor locations. The anchor decoder is then trained on frozen AFNet features to predict coordinates and existence scores. (b) At inference, the predicted scores guide anchor selection, while the observed contour determines anchor order and provides boundary samples for curve fitting. (c) Full-image reconstruction: reconstruct component paths and assemble them into an SVG.

3.2 Anchor-set decoding

Reading anchor proposals from features. The predicted anchor field supports direct peak extraction. However, the number and locations of extracted anchors depend on response thresholds and the nonmaximum-suppression and centroid windows. We therefore use a three-layer attention-based decoder to predict anchor coordinates and existence scores from frozen AFNet features, providing a learned criterion for candidate selection. Intermediate features supply spatial detail, while pooled and bottleneck features supply global context. The field-output head is unused during decoder training and inference. We use this decoder throughout reconstruction and evaluate alternative readouts in Section 4.4.

The decoder maintains learned queries, each paired with a candidate coordinate. Each layer samples fine features around these coordinates, exchanges information among queries through self-attention, and reads global context through cross-attention. Prediction heads then update the normalized coordinate $r_j \in [0,1]^2$ and existence score $p_j \in [0,1]$ for query j. Supplementary Figure 10 details the feature connections and decoder architecture.

Supervising anchor presence and position. A closed contour has no preferred starting anchor or traversal direction. We therefore supervise candidate anchors as a set, without assigning query indices to a prescribed contour order. Let $\bar{a}_i^*$ denote reference anchor a_i^* normalized to $[0,1]^2$. Bipartite matching uses coordinate L_1 distance and focal classification cost to assign each reference anchor to a query $\pi(i)$. Matched queries receive existence label $y_j = 1$, and unmatched queries receive $y_j = 0$. At each layer, we optimize

$$\mathcal{L}_{\text{set}} = \mathcal{L}_{\text{focal}}(p, y) + 25\,\mathcal{L}_{\text{SmoothL1}}\left(\{r_{\pi(i)}\}, \{\bar{a}_i^*\}\right), \tag{3}$$

where $p = (p_j)$ and $y = (y_j)$. The focal loss supervises existence for all queries, while the Smooth-L_1 loss supervises coordinates for matched queries. Losses are normalized by the target count and averaged across layers. At inference, candidates with final scores $p_j \geq \tau$ are retained, where τ is the selection threshold, allowing the initial anchor count to vary across inputs.

3.3 Boundary-constrained curve reconstruction

From anchors to fitted curves. We assign selected proposals to distinct locations on the observed contour and order them by boundary arclength. Consecutive anchors define the endpoints and boundary span of each curve. For segment i, we fix endpoints a_i and a_{i+1} and fit a cubic Bézier curve $B_i(t; \mathbf{H}_i)$, where $\mathbf{H}_i$ collects its two inner control points. The curve parameterization is given in Supplementary Eq. (5).

We fit each segment to boundary samples x_{im} at fixed curve parameters t_{im}, using coordinates centered at the midpoint of its endpoints:

$$\widehat{\mathbf{H}}_i = \arg\min_{\mathbf{H}_i} \sum_m \|B_i(t_{im}; \mathbf{H}_i) - x_{im}\|_2^2 + \lambda_i \|\mathbf{H}_i\|_F^2, \tag{4}$$

where $\lambda_i > 0$ is the regularization weight and $\|\cdot\|_F$ denotes the Frobenius norm.

For single-path reconstruction, failed geometric checks or insufficient IoU with the observed input lead us to lower the selection threshold, including more anchor proposals in curve reconstruction.

Removing redundant anchors. After obtaining a valid path, we check for redundant intermediate anchors. For consecutive anchors a, b, c, we refit one curve from a to c against their combined boundary span. We remove b only if the replacement passes sampled self-intersection checks and satisfies fixed tolerances on raster fidelity and geometric deviation.

For full-image reconstruction, an AdaVec-based frontend extracts component contours (Zhao et al., 2025). We apply the path reconstruction procedure to each component, map the resulting paths back to the image canvas, and assemble them with their fills and layer order before global optimization.

4 Experiments

4.1 Experimental setup

Data and training. We train AnchorFlow on 31,950 paths extracted from 15,996 SVG files. Sources include SVG data from StarVector (Rodriguez et al., 2025), Fluent Emoji (Microsoft, 2026), Noto Emoji (Google Fonts, 2026), Game-icons, Health Icons, Material Design Icons, Font Awesome, and Natural Earth. Training uses clean and perturbed contour views with anchor annotations from the source SVGs. Both training stages use a single NVIDIA A100 80GB GPU, taking approximately 9.6 hours for AFNet and 5.0 hours for the anchor decoder (14.6 hours total).

Single-path evaluation uses 2,400 paths from ColorSVG (Chen & Pan, 2025), Bootstrap Icons (Bootstrap Team, 2025), and STIX (STIX Fonts Project, 2021). For each path, we evaluate a clean raster and one fixed boundary-perturbed input against the same clean reference. Full-image evaluation randomly samples 100 images each from Noto Emoji, Fluent Emoji, and HeisenVec (Zini et al., 2025). Source SVG files used to extract training paths are excluded from both single-path and full-image evaluation.

Baselines. We compare tracing methods Potrace (Selinger, 2003), VTracer (Vision Cortex, 2024), and AutoTrace (AutoTrace Project, 2024); optimization-based methods DiffVG (Li et al., 2020) and LIVE (Ma et al., 2022); SVG generators OmniSVG (Yang et al., 2025) and InternSVG (Wang et al., 2026); and AdaVec (Zhao et al., 2025) for adaptive parameterization.

Metrics. For single paths, we report region IoU, symmetric boundary Chamfer distance (CD, in pixels), and anchor count. Count MAE measures mean absolute anchor-count error relative to the source SVG, while GOSPA (Rahmathullah et al., 2017) measures the discrepancy between reconstructed and source-SVG anchor sets. For full images, RGB MSE and PSNR measure pixel-level

Table 1: Clean and perturbed reconstruction of 2,400 paths. Best results are bold; second-best are underlined. N is the path budget, T is the maximum output-token count.

Input	Type	Method	Comp. (%) ↑	IoU (%) ↑	CD (px) ↓	Count Anchors↓	Count MAE ↓	GOSPA↓	Time (s) ↓
Clean	Tracing	Potrace	100.0	99.168	0.109	17.88	6.745	0.0562	0.0146
		VTracer	100.0	97.664	0.314	36.83	25.243	0.0774	**0.0036**
		AutoTrace	100.0	98.073	0.257	12.97	3.565	0.0501	0.0711
	Opt.	DiffVG ($N = 5$)	100.0	89.424	18.738	20.00	10.946	0.0725	24.52
		LIVE ($N = 1$)	100.0	94.930	4.132	16.00	8.305	0.0653	52.31
	Gen.	OmniSVG ($T = 1{,}024$)	86.2	59.657	44.593	22.63	13.194	0.0475	5.47
		InternSVG ($T = 1{,}024$)	94.1	64.897	24.993	14.53	6.642	0.0498	4.77
	Adapt.	AdaVec	100.0	99.254	0.087	21.95	11.340	0.0620	9.65
		Ours	100.0	**99.521**	**0.072**	**9.53**	**2.663**	**0.0254**	0.836
Perturbed	Tracing	Potrace	100.0	97.523	0.334	60.51	48.758	0.1023	0.266
		VTracer	100.0	96.456	0.495	96.63	84.837	0.1307	**0.252**
		AutoTrace	100.0	96.596	0.468	18.83	8.228	0.0607	0.475
	Opt.	DiffVG ($N = 5$)	100.0	87.976	19.397	20.00	10.946	0.0734	53.65
		LIVE ($N = 1$)	100.0	63.575	83.903	16.00	8.305	0.0688	83.13
	Gen.	OmniSVG ($T = 1{,}024$)	14.5	8.363	222.394	159.93	32.226	0.0645	45.76
		InternSVG ($T = 1{,}024$)	20.6	6.886	217.258	52.70	19.317	0.0569	42.56
	Adapt.	AdaVec	97.3	94.992	7.144	36.89	25.195	0.0813	11.07
		Ours	100.0	**97.687**	**0.317**	**12.55**	**3.470**	**0.0376**	2.13

fidelity, LPIPS (Zhang et al., 2018) measures perceptual differences, and SSIM (Wang et al., 2004) measures structural similarity. Comp. denotes the percentage of completed outputs.

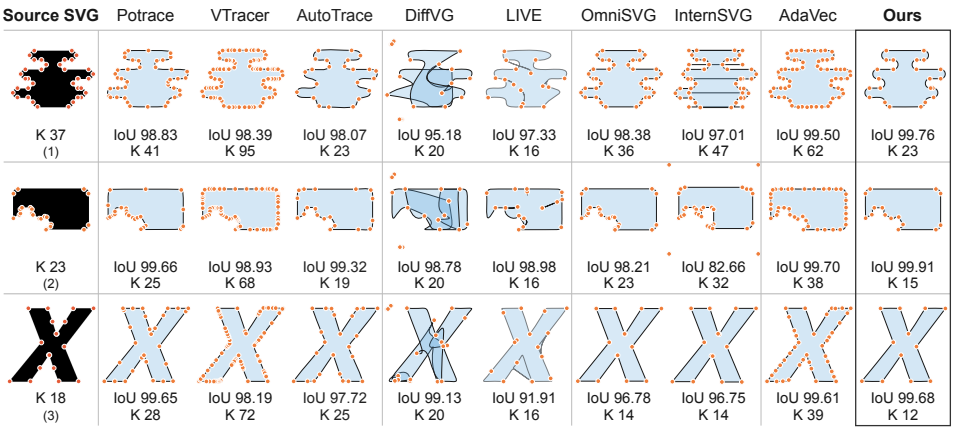

Figure 3: **Clean single-path reconstruction.** The first column shows black input raster shapes with orange source-SVG anchors. IoU (%) and anchor count K appear below each reconstruction.

4.2 Single-Path Reconstruction

Clean reconstruction. On clean inputs, AnchorFlow achieves the highest mean IoU (99.521%) and the lowest CD (0.072 pixels), anchor count (9.53), count MAE, and GOSPA (Table 1). It improves fidelity with 56.6% fewer anchors than AdaVec; even the most compact tracer, AutoTrace, uses 12.97 anchors.

Figure 3 illustrates the reconstructed control structures. Example (1) preserves indentations and straight-to-curve transitions with sparser anchors than VTracer and AdaVec. Example (2) captures the curved profile at 99.91% IoU with 15 anchors, versus 38 for AdaVec. In example (3), 12 anchors capture the X-shaped contour's straight edges and sharp corners at 99.68% IoU, versus 28 for Potrace

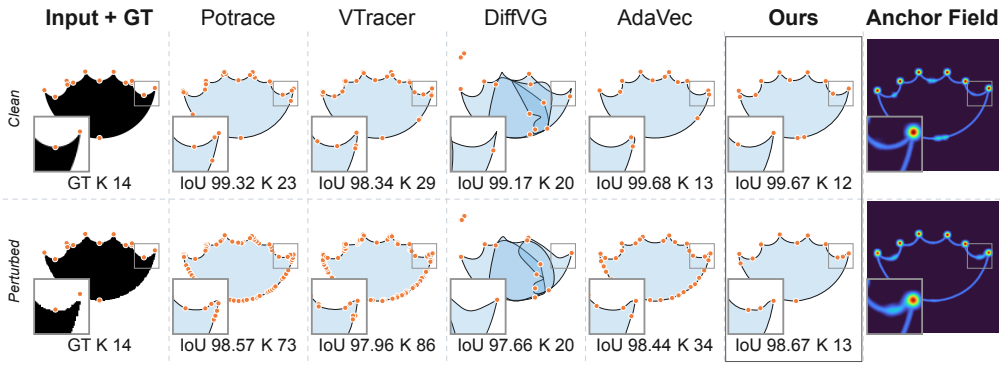

Figure 4: **Reconstruction under boundary perturbations.** Clean and perturbed inputs and reconstructions appear in the top and bottom rows, respectively. AFNet fields on the right.

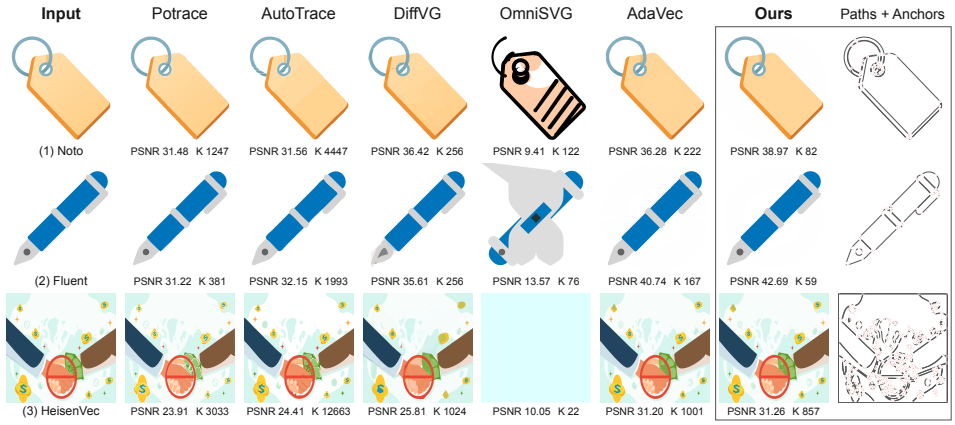

Figure 5: **Full-image reconstruction.** Qualitative comparisons on Noto Emoji, Fluent Emoji, and HeisenVec. The last column visualizes our SVG paths and anchors.

and 39 for AdaVec. Together, these results show that compact control structures can preserve both smooth contours and sharp local features.

Robustness to boundary perturbations. Segmentation, resizing, and rasterization can introduce boundary irregularities. We therefore test robustness using lower-resolution rasterization and weak boundary displacement. In the perturbed results of Table 1, AnchorFlow maintains 97.687% mean IoU and a CD of 0.317 pixels with 12.55 anchors on average. Relative to clean inputs, its IoU decreases by 1.834 percentage points, versus 4.262 for AdaVec, while its anchor count grows by 31.7%, versus 68.1%. Potrace achieves a comparable IoU of 97.523%, but its anchor count grows by 238.4%. LIVE falls from 94.930% to 63.575% mean IoU, indicating sensitivity to these perturbations. Thus, the fidelity–compactness advantage persists under these perturbations: AnchorFlow preserves the underlying shape without the large anchor growth observed in the tracing baselines.

4.3 Full-image reconstruction

Applied to full-image components, AnchorFlow uses the fewest anchors on all three datasets and leads on the four fidelity metrics, except for second-best MSE on Fluent Emoji and LPIPS on HeisenVec (Table 2). Anchor counts on Noto and Fluent are 31.1% and 55.4% below the next most compact baseline. Figure 5 shows higher PSNR with substantially fewer anchors than Potrace and AutoTrace; OmniSVG exhibits larger shape and layout deviations. In example (2), AnchorFlow preserves the pointed outline and circular feature where DiffVG distorts the pen tip. Our outputs remain visually close to AdaVec's with fewer anchors, including the complex scene in example (3).

Table 2: **Full-image reconstruction.** Quantitative comparison of representation compactness, reconstruction quality, and runtime on Noto Emoji, Fluent Emoji, and HeisenVec (100 images each).

Dataset	Type	Method	Anchors↓	MSE↓	LPIPS↓	PSNR↑	SSIM↑	Time (s)
Noto Emoji	Tracing	Potrace (color)	1732.45	0.003768	0.09187	25.34	0.9131	**0.22**
		AutoTrace	6262.27	0.008919	0.06353	26.95	0.9479	0.33
	Opt.	DiffVG ($N = 64$)	256.00	0.001911	0.07013	30.00	0.9550	41.47
	Gen.	OmniSVG ($T = 4{,}096$)	603.93	0.139472	0.49974	10.20	0.7179	69.59
	Adapt.	AdaVec	281.13	0.001287	0.02918	34.66	0.9787	31.17
		Ours	**176.49**	**0.000484**	**0.01298**	**37.32**	**0.9872**	37.09
Fluent Emoji	Tracing	Potrace (color)	318.98	0.001478	0.03161	29.00	0.9718	**0.19**
		AutoTrace	4167.44	0.010476	0.07829	26.83	0.9635	0.28
	Opt.	DiffVG ($N = 64$)	256.00	0.000718	0.02757	36.12	0.9856	40.88
	Gen.	OmniSVG ($T = 4{,}096$)	298.08	0.071542	0.29301	15.10	0.8349	33.11
	Adapt.	AdaVec	157.49	**0.000180**	0.00930	39.94	0.9949	24.89
		Ours	**70.27**	0.000195	**0.00558**	**41.89**	**0.9956**	41.21
HeisenVec	Tracing	Potrace (color)	3314.02	0.004243	**0.06673**	24.51	0.9180	**0.37**
		AutoTrace	10992.80	0.006089	0.07671	23.65	0.9179	0.38
	Opt.	DiffVG ($N = 256$)	1024.00	0.002824	0.10758	26.14	0.9163	148.01
	Gen.	OmniSVG ($T = 8{,}192$)	1312.51	0.126347	0.60392	10.85	0.6887	186.25
	Adapt.	AdaVec	926.84	0.002152	0.07899	27.82	0.9408	79.96
		Ours	**892.11**	**0.001779**	0.07206	**28.25**	**0.9449**	172.64

Table 3: **Readout and field pretraining.** Means over 2,400 clean paths; all outputs valid.

Method	K ↓	IoU↑	GOSPA↓
Field-peak readout	9.872	**0.995679**	0.025616
Joint from scratch	9.670	0.995151	0.025869
Ours	**9.533**	0.995211	**0.025378**

Table 4: **Effect of anchor initialization.** Results on 2,400 clean paths.

Method	Initial K	Final K ↓	IoU↑	GOSPA↓
Potrace	17.877	12.370	0.994684	0.046554
AdaVec	21.945	11.516	0.994856	0.046738
Ours	**11.585**	**9.533**	**0.995211**	**0.025378**

These results show that the compactness and fidelity of the path module carry over to assembled images with multiple components.

4.4 Ablation Study

We examine what the learned representation and anchor placement contribute. Table 3 compares readout and training choices; Tables 4 and 5 control the reconstruction backend and initial count to examine the value of anchor initialization and placement.

Table 5: **Anchor placement at matched counts.** Results on 2,400 clean paths. A: handle fitting with fixed anchors. B: shared refinement, independently restarted from the same initializations.

		A: Handle fitting only		B: Shared refinement		
Method	Initial K	IoU ↑	GOSPA ↓	Final K ↓	IoU ↑	GOSPA ↓
Uniform	13.879	0.977403	0.058960	11.293	0.988979	0.052135
Error-greedy	13.879	0.985210	0.047443	10.478	0.991846	0.040808
Ours	13.879	**0.993993**	**0.030902**	**10.098**	**0.995833**	**0.025928**

Anchor readout and field pretraining. We compare field-peak extraction and the anchor decoder using the same field-pretrained AFNet. In Table 3, field peaks achieve slightly higher IoU, while the decoder uses 3.4% fewer final anchors and has 0.9% lower GOSPA. The field features therefore support accurate reconstruction through either readout, with the decoder favoring a more compact anchor structure.

Under the same total training budget of 120 epochs, we compare our two-stage scheme (60 field-training and 60 decoder-training epochs) with joint AFNet–decoder training from scratch using anchor-set supervision alone. Our scheme yields fewer anchors, slightly higher IoU and lower GOSPA, with shorter recorded training time (14.6 vs. 26.5 hours). Field pretraining thus provides a useful initialization for compact reconstruction within the same epoch budget.

Effect of anchor initialization. We compare initial anchors from Ours, Potrace, and AdaVec under a shared geometric backend. We discard the handles produced by Potrace and AdaVec and apply the same curve reconstruction to the observed contours; Ours retains its standard adaptive anchor selection. AdaVec endpoints are taken after simplification but before point and shape optimization. Table 4 shows that Ours yields the fewest final anchors and closer agreement with source-SVG anchors while maintaining comparable fidelity. These results show that the initial anchors affect the final control structure even after the same geometric refinement.

Initial anchor count also affects reconstruction accuracy. To isolate the effect of anchor position, we compare Ours with uniform arclength sampling (Uniform) and greedy insertion by fitting error (Error-greedy), matching their initial counts to Ours for each input. With anchors fixed and only handles fitted, Ours already achieves higher IoU and lower GOSPA (Table 5, A), showing that its anchor positions enable more accurate curves with the same number of segments. This advantage persists after shared refinement (B): compared with Error-greedy, Ours improves mean IoU by 0.40 percentage points, with 3.6% fewer final anchors and 36.5% lower GOSPA. These results show that the anchor positions predicted by Ours make more effective use of the initial anchor budget and lead to more accurate, compact curves after refinement.

4.5 Editing use-case study

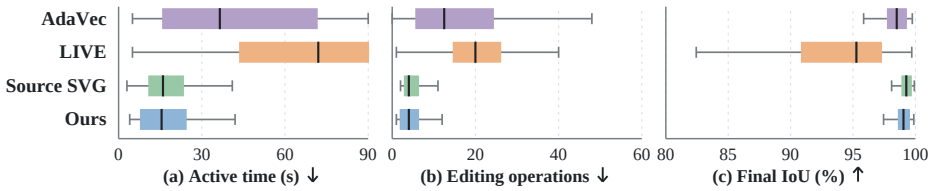

Figure 6: **User study on four SVG editing tasks.** Eighteen participants perform four editing tasks under four conditions: AdaVec, AnchorFlow, LIVE, and the source SVG as a reference.

To examine whether the reconstructed controls support practical editing, we evaluate four distinct editing tasks with 18 participants under four conditions: native AdaVec (Zhao et al., 2025), AnchorFlow, LIVE (Ma et al., 2022), and the source SVG as a reference. Across the four tasks, AnchorFlow achieves editing performance close to the source-SVG reference, requiring a similar median active editing time (15.5 vs. 16.0 seconds) and the same median number of geometric actions (4), while reaching a comparable final mask IoU (99.04% vs. 99.27%). Compared with AdaVec and LIVE, AnchorFlow requires less time and fewer actions while achieving higher final target IoU on every task. The compact reconstructed controls therefore translate into reduced editing effort while preserving target accuracy on these tasks.

5 Conclusion and Limitations

We presented AnchorFlow, which learns anchor placement from designer-authored SVGs through a sparse field encoding contour geometry and reference anchors. An anchor decoder predicts explicit anchors for boundary-constrained curve fitting. Experiments show faithful, compact reconstruction on single paths and full images, alongside efficient editing on the selected tasks. The method remains sensitive to errors in component decomposition and anchor placement, which can omit components or distort fine details, and is slower than the evaluated tracing methods.

AI USE STATEMENT

Generative AI tools assisted with experiment-design feedback, code implementation and review, literature and format checks, translation, and manuscript drafting and editing. Numerical claims were checked against saved experimental outputs. The authors take responsibility for the manuscript and results.

REPRODUCIBILITY STATEMENT

Sections A.4 and B of the Supplementary Material specify the architecture, training, reconstruction, evaluation, and ablation protocols. The accompanying code provides pretrained inference weights, single-path reconstruction and evaluation on 2,400 clean inputs, and six full-image examples. Its reproduction scope is detailed in the README.

REFERENCES

Anthropic. Claude Fable 5.1 & Claude Mythos 5.1 System Card, September 2026a. URL https://www.anthropic.com/claude-fable-5-1-mythos-5-1-system-card.

Anthropic. Claude Opus 5 System Card, July 2026b. URL https://www.anthropic.com/claude-opus-5-system-card.

AutoTrace Project. AutoTrace: Bitmap to vector graphics converter. https://autotrace.sourceforge.net/, 2024. Open source software.

Bootstrap Team. Bootstrap Icons. https://github.com/twbs/icons/releases/tag/v1.13.1, 2025. Version 1.13.1, MIT license.

Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. DeepSVG: A hierarchical generative network for vector graphics animation. In *Advances in Neural Information Processing Systems*, volume 33, pp. 16351–16361, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/bcf9d6bd14a2095866ce8c950b702341-Abstract.html.

Zehao Chen and Rong Pan. SVGBuilder: Component-based colored svg generation with text-guided autoregressive transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 2358–2366, 2025. doi: 10.1609/aaai.v39i3.32236. URL https://ojs.aaai.org/index.php/AAAI/article/view/32236.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the Limits of KV Cache Compression, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash/blob/main/DeepSeek_V41_Tech_Report.pdf.

Google DeepMind. Gemini 3.8 Flash Model Card, September 2026. URL https://deepmind.google/models/model-cards/gemini-3-8-flash/.

Google Fonts. Noto Emoji. https://github.com/googlefonts/noto-emoji, 2026. Open source emoji library with vector SVG and PNG assets.

Or Hirschorn, Amir Jevnisek, and Shai Avidan. Optimize & reduce: A top-down approach for image vectorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 2148–2156, 2024. doi: 10.1609/aaai.v38i3.27987. URL https://ojs.aaai.org/index.php/AAAI/article/view/27987.

Teng Hu, Ran Yi, Baihong Qian, Jiangning Zhang, Paul L. Rosin, and Yu-Kun Lai. Supersvg: Superpixel-based scalable vector graphics synthesis. *arXiv preprint arXiv:2406.09794*, 2024.

Jaerin Lee, Kanggeon Lee, and Kyoung Mu Lee. Vector scaffolding: Inter-scale orchestration for differentiable image vectorization. In *European Conference on Computer Vision*, 2026. URL https://arxiv.org/abs/2605.11913.

Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics*, 39(6):193:1–193:15, 2020. doi: 10.1145/3414685.3417871. URL https://people.csail.mit.edu/tzumao/diffvg/.

Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. Towards layer-wise image vectorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16314–16323, 2022. doi: 10.1109/CVPR52688.2022.01583. URL https://openaccess.thecvf.com/content/CVPR2022/html/Ma_Towards_Layer-Wise_Image_Vectorization_CVPR_2022_paper.html.

Microsoft. Fluent Emoji. https://github.com/microsoft/fluentui-emoji, 2026. Open source emoji collection, MIT license.

OpenAI. GPT-5.6 System Card, July 2026a. URL https://deploymentsafety.openai.com/gpt-5-6.

OpenAI. GPT-6 Astra System Card, September 2026b. URL https://deploymentsafety.openai.com/gpt-6-astra.

Abu Sajana Rahmathullah, Ángel F. García-Fernández, and Lennart Svensson. Generalized optimal sub-pattern assignment metric. In *20th International Conference on Information Fusion*, 2017. URL https://arxiv.org/abs/1601.05585.

Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images and text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16175–16186, 2025. doi: 10.1109/CVPR52734.2025.01508. URL https://openaccess.thecvf.com/content/CVPR2025/html/Rodriguez_StarVector_Generating_Scalable_Vector_Graphics_Code_from_Images_and_Text_CVPR_2025_paper.html.

Peter Selinger. Potrace: A polygon-based tracing algorithm, September 2003. URL https://www.mathstat.dal.ca/~selinger/potrace/potrace.pdf. Technical report.

STIX Fonts Project. STIX Two Math. https://github.com/stipub/stixfonts/releases/tag/v2.13b171, 2021. Version 2.13 b171, SIL Open Font License 1.1.

Vision Cortex. VTracer: Raster to vector graphics converter. https://github.com/visioncortex/vtracer, 2024. Open source software.

Haomin Wang, Jinhui Yin, Qi Wei, Wenguang Zeng, Lixin Gu, Shenglong Ye, Zhangwei Gao, Yaohui Wang, Yanting Zhang, Yuanqi Li, Yanwen Guo, Wenhai Wang, Kai Chen, Yu Qiao, and Hongjie Zhang. InternSVG: Towards unified svg tasks with multimodal large language models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=YxqnNNs3sf.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004.

Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Jiaxu Zhang, Liao Wang, Gang Yu, Xingjun Ma, and Yu-Gang Jiang. Omnisvg: A unified scalable vector graphics generation model. *arXiv preprint arXiv:2504.06263*, 2025.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Kaibo Zhao, Liang Bao, Yufei Li, Xu Su, Ke Zhang, and Xiaotian Qiao. Less is more: Efficient image vectorization with adaptive parameterization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18166–18175, 2025. doi: 10.1109/CVPR52734.2025.01693. URL https://openaccess.thecvf.com/content/CVPR2025/html/Zhao_Less_is_More_Efficient_Image_Vectorization_with_Adaptive_Parameterization_CVPR_2025_paper.html.

Hengyu Zhou, Hui Zhang, and Bin Wang. Segmentation-guided layer-wise image vectorization with gradient fills. *arXiv preprint arXiv:2408.15741*, 2024.

Haokun Zhu, Juang Ian Chong, Teng Hu, Ran Yi, Yu-Kun Lai, and Paul L. Rosin. Samvg: A multi-stage image vectorization model with the segment-anything model. *arXiv preprint arXiv:2311.05276*, 2023.

Leonardo Zini, Elia Frigieri, Sebastiano Aloscari, and Lorenzo Baraldi. vHector and HeisenVec: Scalable vector graphics generation through large language models. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/20b5750c81b8ef055bd99b4915fe29a3-Paper-Datasets_and_Benchmarks_Track.pdf.

SUPPLEMENTARY MATERIAL

A SUPPLEMENTARY EXPERIMENTS AND ANALYSES

A.1 FAILURE CASE STUDY

Figure 7 illustrates two failure modes. In (a), errors in the component decomposition omit or merge foreground regions, leaving details missing from the assembled image. In (b), the reconstructed rake preserves the overall silhouette, but the widths and local curvature of individual teeth deviate from the reference. The first case concerns the decomposition frontend; the second shows that local anchor placement and curve fitting can still distort fine geometry.

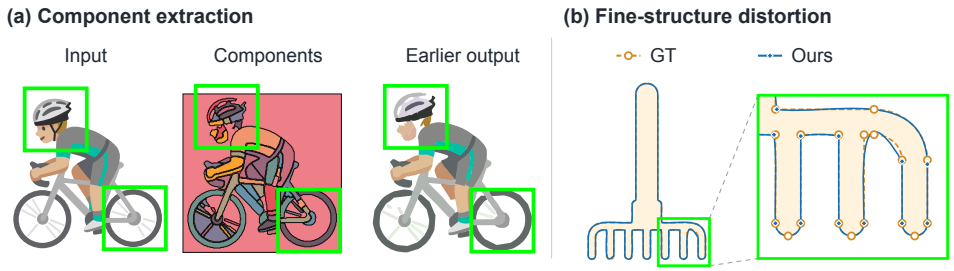

Figure 7: **Failure cases.** (a) Component decomposition errors affecting full-image reconstruction. (b) Fine-detail distortion in single-path reconstruction.

A.2 COMPARISON WITH GENERAL-PURPOSE MODELS

We randomly sample 100 clean paths from the 2,400-path benchmark and compare GPT-6 Astra (OpenAI, 2026b), GPT-5.6 Sol (OpenAI, 2026a), Claude Fable 5.1 (Anthropic, 2026a), Claude Opus 5 (Anthropic, 2026b), DeepSeek-V4.1-Flash (DeepSeek-AI, 2026), and Gemini 3.8 Flash (Google DeepMind, 2026). All models receive the same inputs and produce one candidate per input.

ANCHORFLOW achieves 99.617% mean IoU with 9.74 anchors, compared with 97.778% and 97.239% IoU for Astra and Sol. It also has lower CD and GOSPA than all six baselines (Figure 8). Model settings and measurable-output counts are given in Section B.3.

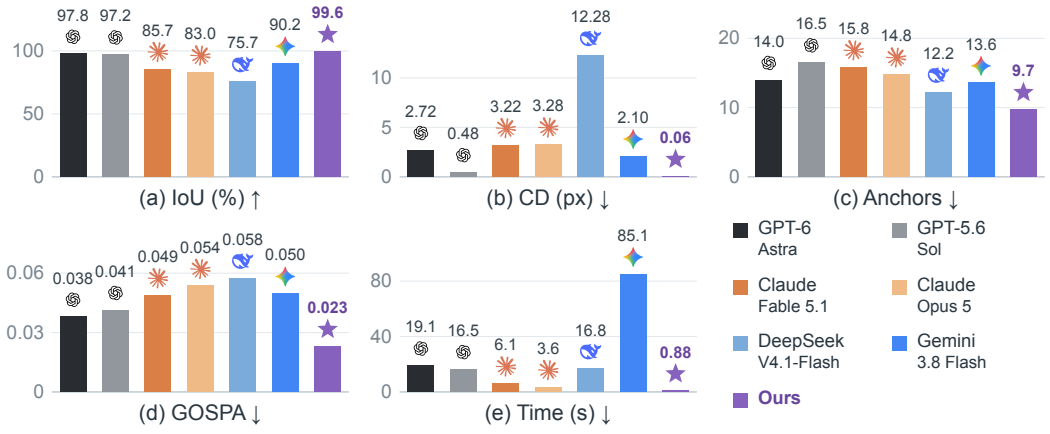

Figure 8: **Comparison with general-purpose models.** Mean reconstruction quality, anchor structure, and recorded latency on 100 clean paths, with one candidate per input. Quality means include all attempts; anchor counts use measurable outputs.

A.3 EDITING STUDY

Eighteen participants contributed 288 formal attempts (72 per condition) after guided practice. Figure 9 shows four selected tasks with common targets and initial SVGs from AdaVec, LIVE, ANCHORFLOW, or the source-SVG structural control. Participants saw anonymous method labels, with presentation order counterbalanced across the four tasks so that each condition appeared in different positions. Each attempt had a 90-second active editing limit.

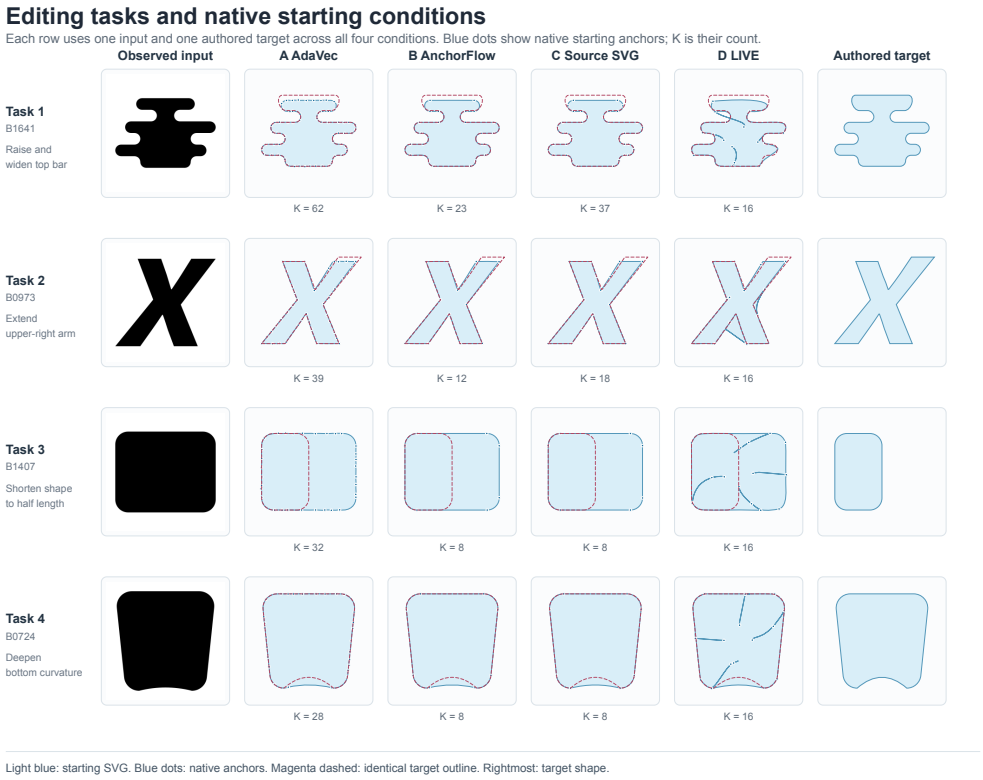

Figure 9: **Editing tasks and starting SVGs.** Blue dots mark the K initial anchors, and magenta dashed contours show targets.

Active time and all summaries include finished, timed-out, and abandoned attempts. Actions count insertions, deletions, and moves.

Table 6: **Per-task editing results.** Medians over 18 attempts per cell, including all completion statuses. Top line: active time (s) / actions. Bottom line: initial → final mask IoU (%).

Task	AdaVec	LIVE	Source SVG	ANCHORFLOW
Top bar	54.0 / 16.0 90.86 → 98.57	88.5 / 20.0 89.53 → 95.27	21.0 / 6.5 91.22 → 99.20	17.5 / 3.0 91.03 → 99.00
X arm	16.0 / 6.5 94.33 → 98.18	55.0 / 17.5 87.43 → 93.20	13.0 / 4.0 94.58 → 99.18	8.0 / 2.5 94.41 → 98.76
Shorten	74.0 / 25.5 49.07 → 98.19	86.0 / 25.5 49.01 → 91.21	18.5 / 5.0 49.07 → 99.17	22.0 / 6.0 49.05 → 99.12
Bottom curve	18.5 / 5.0 96.59 → 99.44	55.5 / 16.0 96.51 → 98.96	10.5 / 3.0 96.72 → 99.75	15.5 / 4.0 96.67 → 99.58

A.4 ADDITIONAL ABLATION DETAILS

Readout and training controls. Field-peak readout and ours share the field-pretrained AFNet; joint training from scratch learns AFNet and the anchor decoder using anchor-set supervision alone. All use the same inputs and base fitting and merging procedures. Field peaks use response floor 0.16, relative threshold 10^{-6}, a 7×7 nonmaximum-suppression window and an 11×11 centroid window. Joint training runs for 120 epochs, matching our 60+60-epoch budget (Table 8); its checkpoint is selected at epoch 110.

For this ablation, failed query-based reconstructions are retried at score thresholds 0.2 and 0.1, followed, if needed, by local handle repair that preserves anchor positions, order, and count. Only one joint-from-scratch output required this recovery.

Anchor placement at matched initial counts. For each of the 2,400 clean paths, all three initializers receive the count K_i from our fixed-$\tau = 0.2$ prediction, including the minimum-three-anchor rule and distinct contour assignment. Uniform uses equal arclength spacing. Error-greedy starts with three equally spaced anchors and repeatedly inserts the unused contour sample with largest fitting error until it reaches K_i. This tests placement conditional on the learned count budget. The adaptive selection used in Table 4 explains its different mean initial count (11.585 versus 13.879 here).

All methods share the observed contour, coordinate system, anchor assignment, ordering, and handle fitter. Stage A fixes anchors and fits handles only, with no insertion, deletion, merging, or retry. Stage B independently restarts from the same initializations and applies common refinement, with identical insertion-budget rules, merge tolerances, and stopping conditions; method-specific retries are disabled. Budgets use the original K_i. Stage A yields 2,399/2,392/2,400 valid outputs for Learned/Uniform/Error-greedy; Stage B yields 2,400 for each. All inputs enter the quality means, with failures assigned zero IoU and empty anchor sets for GOSPA. Results are reported in Table 5.

Ours versus fixed-threshold selection. Table 7 evaluates whether our anchor-selection strategy produces more compact reconstructions than a fixed threshold under the same reconstruction pipeline. Ours is compared with fixed $\tau = 0.2$ using the same checkpoint, cached predictions, and fitting and merging procedures. Ours begins at $\tau = 0.4$ and retries at 0.2 if reconstruction checks fail or pre-merge observed-input IoU is below 0.99. The minimum-three-anchor rule applies only at 0.2, and insertion budgets are computed per attempt.

Table 7: **Quality and compactness under two selection policies.** Each policy produces valid outputs for all 2,400 clean and all 2,400 paired perturbed inputs. K is the final anchor count.

	Clean		Perturbed	
Policy	IoU (%)	K	IoU (%)	K
Fixed 0.2	99.583	10.098	97.696	13.104
Ours	99.521	9.533	97.687	12.550

Compared with fixed $\tau = 0.2$, Ours uses 5.6% and 4.2% fewer final anchors on clean and perturbed inputs, respectively, while maintaining similar mean IoU.

B IMPLEMENTATION AND EVALUATION SETTINGS

Experiment-specific ablation controls are given in Section A.4.

B.1 NETWORK, TRAINING, AND RECONSTRUCTION

Table 8: **Network and training configuration.** Both stages use 31,950 source paths with clean and perturbed contour views, AdamW, and effective batch size 64.

Setting	Field learning	Anchor decoding
Trainable module	AFNet	Anchor decoder; AFNet frozen
Architecture	Base width 64; one input/output channel	3 refinement layers; 4 attention heads
Input / features	512×512 contour	e_1 and d_2 projected to 96 channels at 128×128, summed and normalized
Supervision	Contour-and-anchor field	Matched coordinates and existence at every layer
Epoch budget	60	60
Learning rate	1.6×10^{-4}, then 8×10^{-5}	5×10^{-4}
Schedule	Cosine decay	2-epoch warmup; cosine decay
Selected epoch	59	52
Field targets	Contour: amplitude 0.15, standard deviation 2 px, cutoff 4 px; anchors: standard deviation ≈ 5.82 px, floating-point centers. Targets share the input's coordinate transform.	
Decoder memory	Fine features pooled to 32×32, plus bottleneck tokens and positional embeddings	
Local sampling	3×3 neighborhood with offsets $\{-8, 0, 8\}/512$	
Matching / losses	Coordinate L_1 matching weight 25; focal $\alpha = 0.25$, $\gamma = 2$; Smooth-L_1 transition $1/512$	

Checkpoints are selected using clean and deployment validation. The validation pools contain 2,300 randomly held-out clean examples and are disjoint from both the training and test sets.

Table 9: **Reconstruction settings.** Merge tolerances are measured relative to the path before merging, on the 256×256 evaluation canvas.

Setting	Value / rule
Single-path normalization	Center foreground bounding box; scale longest side to 0.72×512 px
Anchor assignment	Distinct observed-contour samples, ordered by arclength
Handle fitting	128 samples per span; regularized least squares (Eq. (4)); 3 parameter-update/refitting steps; handle length capped at 1.5 times endpoint distance
Single-path selection	$\tau = 0.4$; retry at 0.2 on fitting failure or pre-merge observed-input IoU < 0.99; supplement to at least 3 anchors only in the 0.2 branch
Full-image selection	To account for possible rasterization artifacts when small components are enlarged, the accompanying code first tries $\tau = 0.85$ when the component-mask area is below 1,024 pixels and the contour bounding-box span is below 64 pixels on the original canvas. Accept a valid candidate; otherwise compare candidates at $\tau = 0.4$ and 0.2.
Anchor insertion	Insert at high-error boundary locations and refit; budget $\min(4, \max(1, \lceil 0.25K_0 \rceil))$, with K_0 after contour assignment for that attempt
Merge acceptance	Preserve sampled simplicity; IoU drop ≤ 0.0005, CD increase ≤ 0.01 px, sampled arc deviation ≤ 0.5 px
Full-image assembly	Local handle refitting for residual self-intersections; foreground regions on white; 101 global updates, geometric-loss weight 0.1

B.2 NETWORK ARCHITECTURE AND CURVE PARAMETERIZATION

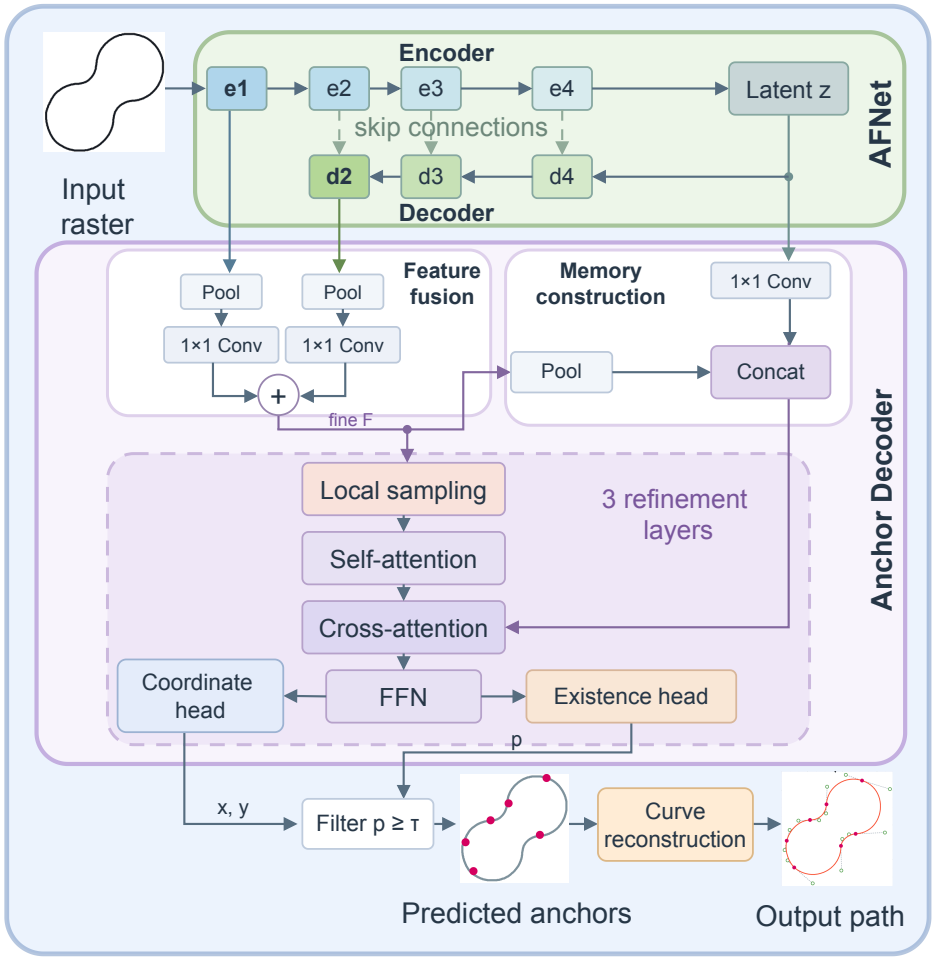

Figure 10: **Network architecture.** AFNet provides encoder, decoder and bottleneck features. Pooled and projected features from e_1 and d_2 are summed to form fine features for local sampling. Pooled fine features and projected bottleneck features form the memory used by cross-attention. Three refinement layers update the anchor queries and coordinates, with an existence head predicting scores for filtering before curve reconstruction.

For ordered anchors $\{a_i\}_{i=1}^{K}$, each segment is parameterized as

$$B_i(t;\mathbf{H}_i) = (1-t)^3 a_i + 3(1-t)^2 t\, h_i^{\text{out}} + 3(1-t)t^2\, h_{i+1}^{\text{in}} + t^3 a_{i+1}, \qquad t \in [0,1], \tag{5}$$

where indices wrap from K to 1, and $\mathbf{H}_i = [h_i^{\text{out}}, h_{i+1}^{\text{in}}]$ contains the two inner control points.

B.3 EVALUATION PROTOCOL

Table 10: **Evaluation data and baseline budgets.** Full-image budgets list Noto / Fluent / HeisenVec in that order. Generative baselines produce one candidate per input.

Setting	Single path	Full image
Test data	1,800 ColorSVG + 226 Bootstrap + 374 STIX paths	100 images each from Noto Emoji, Fluent Emoji, and HeisenVec
Render canvas	256×256, white	512×512 RGB, white
LIVE	16 cubic segments; 500 updates	—
DiffVG	5 paths, 20 total segments; 500 updates	256 / 256 / 1,024 anchors; 500 updates
OmniSVG token cap	1,024	4,096 / 4,096 / 8,192
InternSVG token cap	1,024	—

Paired perturbations. Each source receives one fixed perturbation with raster size 192/224/256, boundary displacement 0.5–0.75 low-resolution pixels, and offset within ± 0.2 pixels. Scoring uses the paired clean reference.

Metrics. IoU measures foreground overlap; MSE uses values in $[0, 1]$; symmetric boundary CD averages both directed nearest-neighbor distances in pixels. Anchors are segment endpoints. For LIVE and DiffVG, anchor count, Count MAE, and GOSPA include every exported path segment regardless of fill color. Count MAE and GOSPA measure agreement with source-SVG anchor layouts, which are reference segmentations rather than unique ground truth. GOSPA uses $p = 2$, $\alpha = 2$, and $c = 0.02$ after normalization by the canvas diagonal. Full-image scoring additionally uses AlexNet v0.1 LPIPS and per-image PSNR, averaged across images.

Failures and coverage. All attempts enter quality means; complexity means use measurable outputs. The supplied single-path evaluator assigns missing or invalid predictions IoU $= 0$, MSE $= 1$, and CD $= 256$. In the ablations in Tables 3–5, failed structures are empty sets for GOSPA. Clean OmniSVG/InternSVG anchor coverage is 1,938/2,400 and 2,257/2,400. Their native-primitives structural diagnostics are excluded from best/second-best highlighting. Full-image OmniSVG anchor coverage is 95/92/85 of 100; truncated generations remain in quality scoring.

General-purpose models. Astra, Sol, Fable, Opus, and DeepSeek use their recorded high settings; Gemini uses the interface default. Measurable-anchor counts are 99/100/99/100/97/99 out of 100, respectively. All attempts enter quality summaries, and GOSPA uses canonicalized SVG anchor sets.

Timing scope. Our times include preprocessing, forward inference, fitting, and merging, excluding one-time model loading. AdaVec times use persistent-process reruns over successfully initialized paths and include point and shape optimization. Other methods retain their recorded execution scopes. Classical tracers run on CPU and learned or optimization methods on A100 GPUs.

C ADDITIONAL QUALITATIVE RESULTS

These curated examples cover varied geometry and scenes; they do not estimate the frequency of particular outcomes.

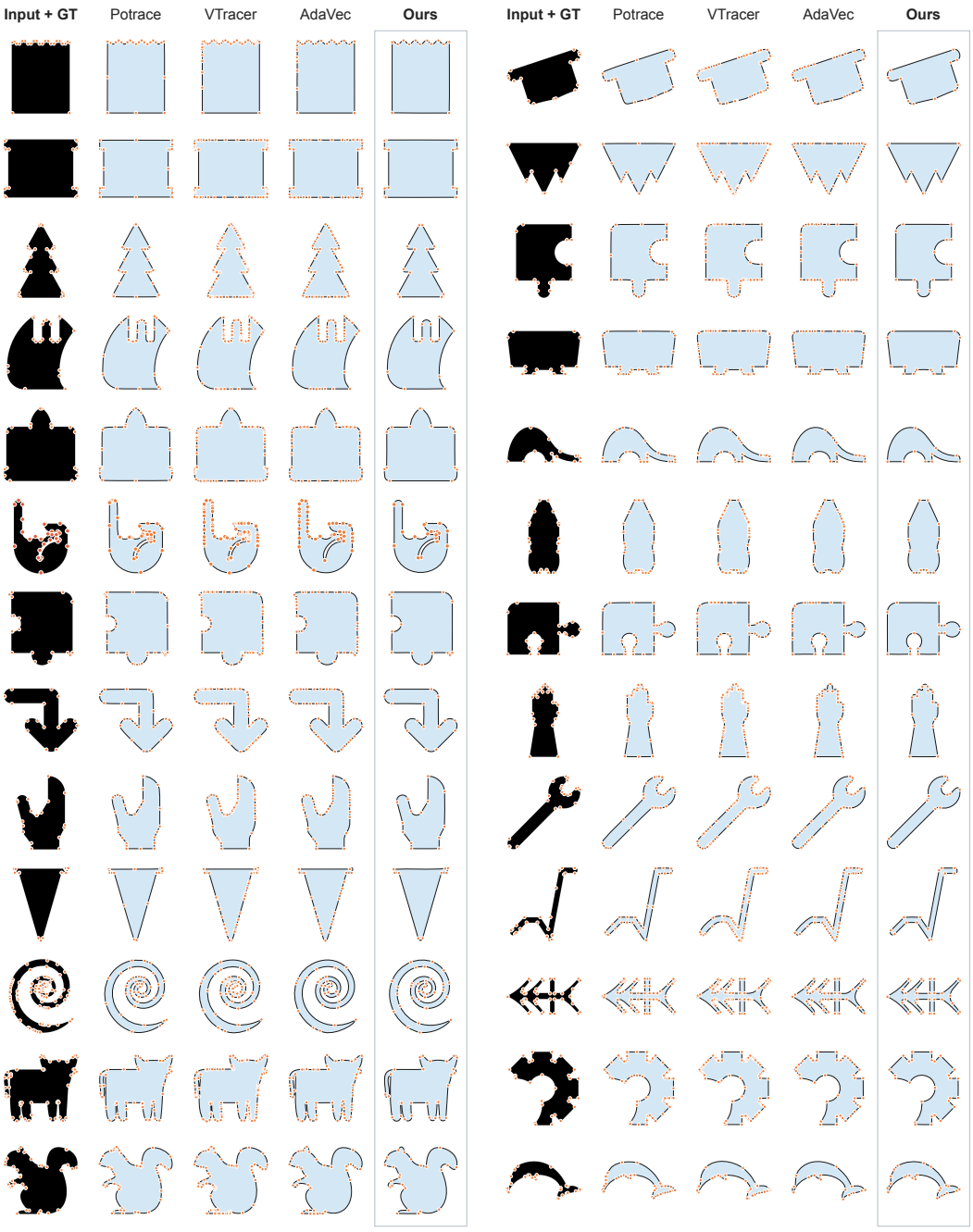

Figure 11: **Clean single-path examples.** Input + GT overlays source-SVG anchors on the raster. Light-blue fills, black paths, and orange native segment endpoints are shown for all methods.



Figure 12: **Noto Emoji and Fluent Emoji examples.** Native colors are retained without anchor overlays. DiffVG uses 256 anchors.

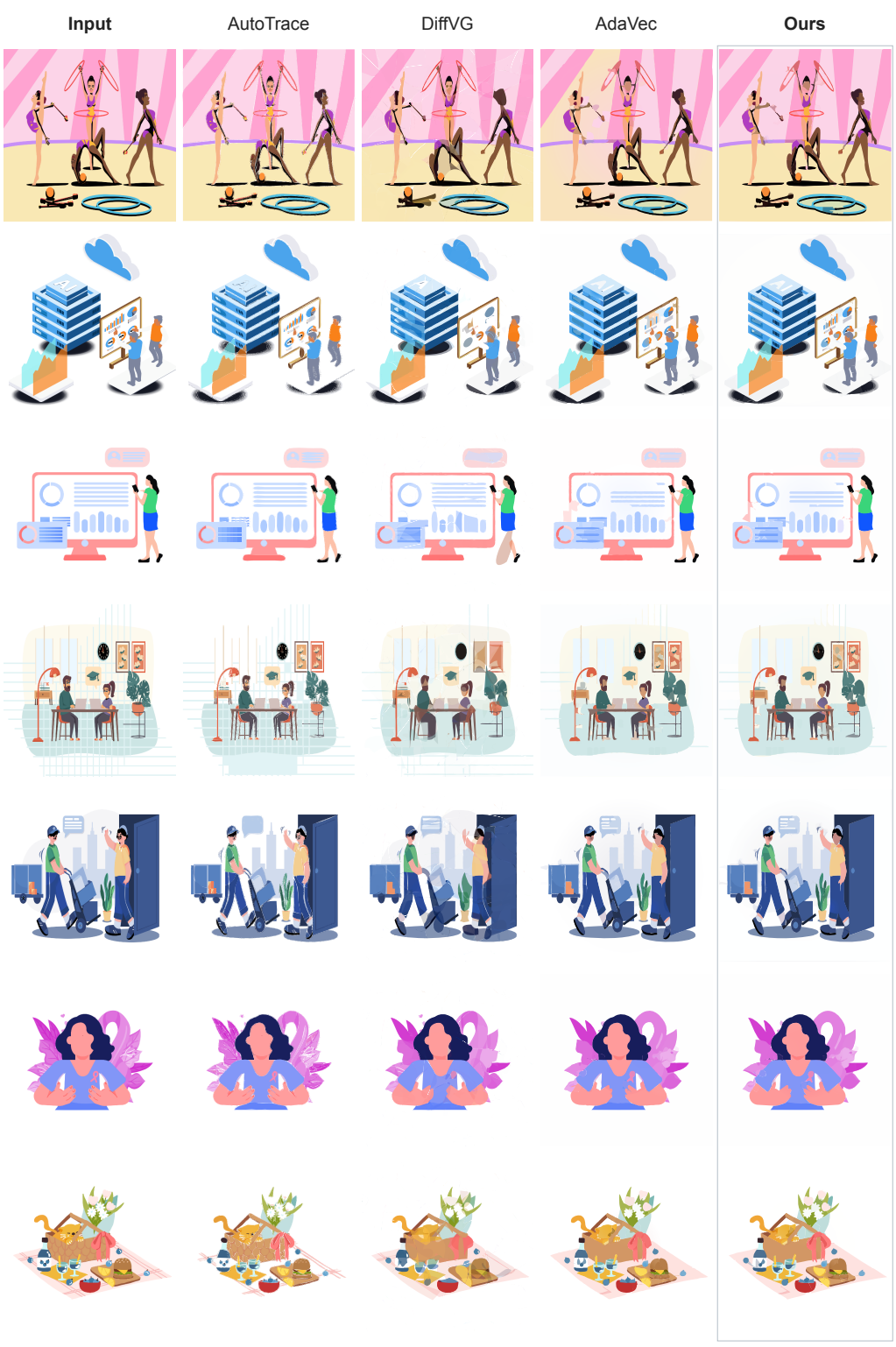

Figure 13: **HeisenVec examples.** Native colors are retained without anchor overlays. DiffVG uses 1,024 anchors.

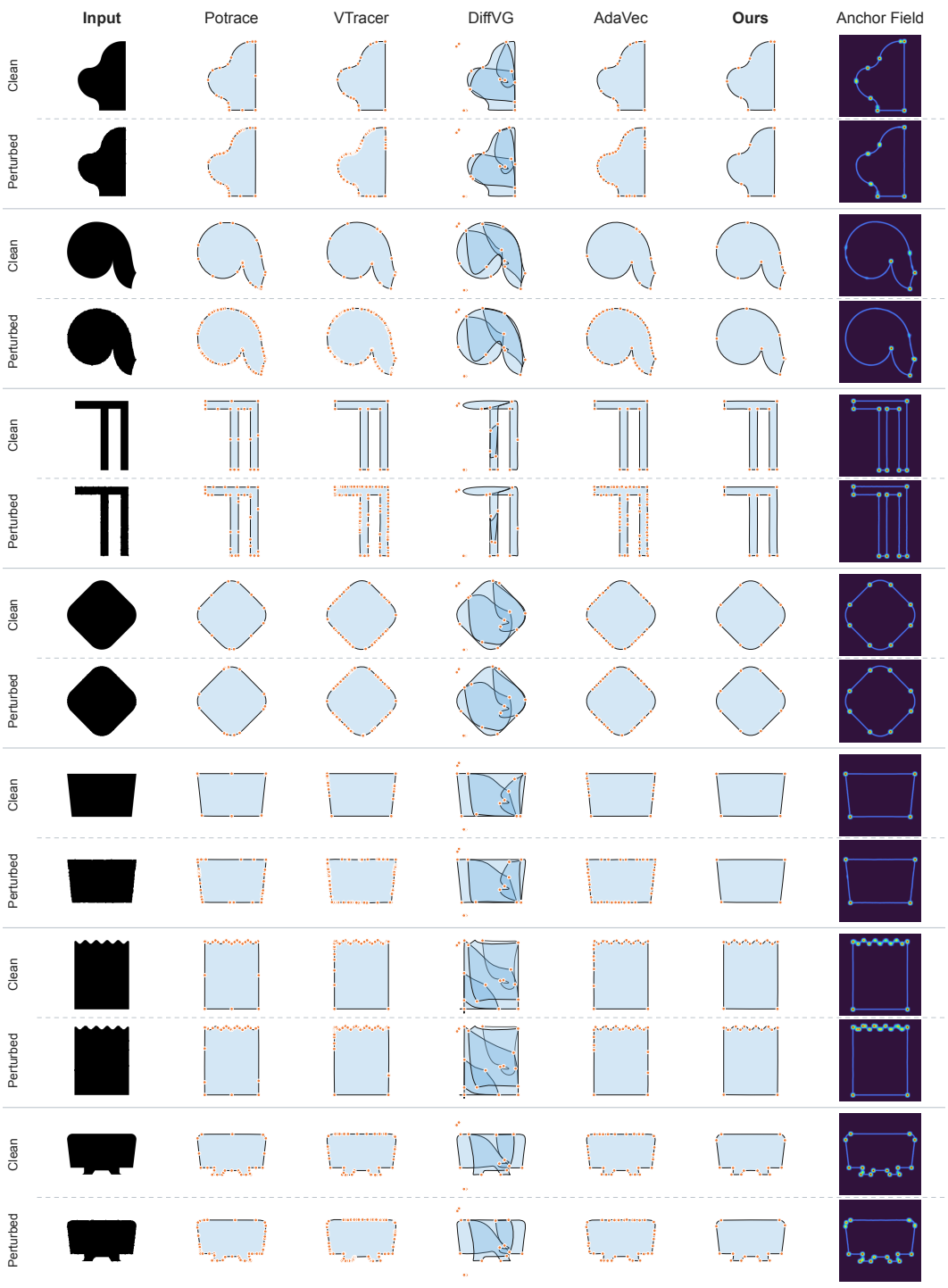

Figure 14: **Paired clean and perturbed inputs.** Orange markers show native segment endpoints. Anchor-field-head responses share a fixed 0–1 color scale; the coordinate decoder reads backbone features rather than this diagnostic output map.