VectorArk: Learning Practical Image Vectorization with Rounded Polygon Representation

Supplementary Material

1. Additional Results

1.1. Outline Comparison on Noisy Inputs

Figure 1 compares the generated outlines on noisy inputs. Our method maintains cleaner geometric structure compared to competing approaches.

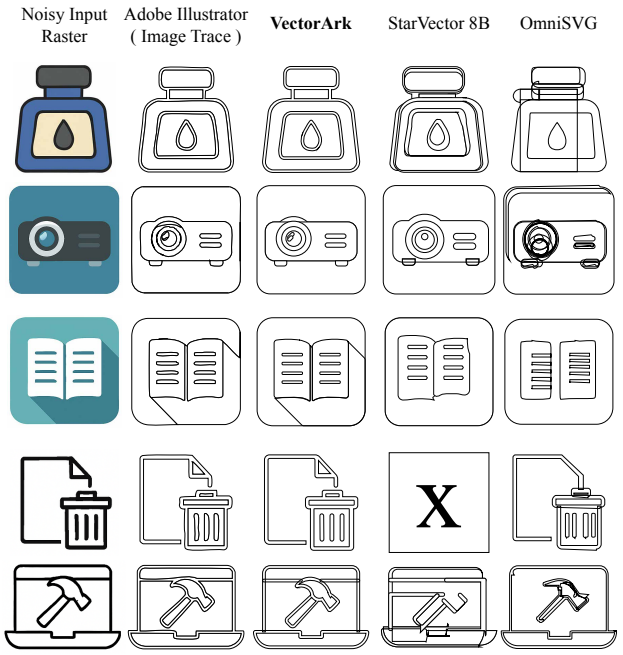


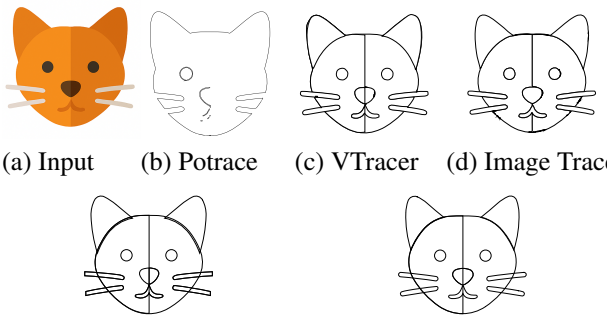
Figure 1. Outline comparison across methods on noisy inputs. *(Zoom in to see geometric artifacts.)*

1.2. Vectorizer Comparison

Our preprocessing pipeline converts a raster image into an outline representation before feeding it to the model. While the main paper uses Adobe Illustrator's Image Trace for this step, we investigated several standard open-source alternatives to assess pipeline robustness and identify a freely available replacement.

Evaluated Vectorizers. We experimented with three widely used vectorizers: Potrace [4], VTracer [2], and the optimization-based method LIVE [1]. As shown in the top row of Figure 2, Potrace produces patchy, often broken outlines with low noise resistance, making it unsuitable for practical use. LIVE requires manually specifying presets and the number of paths, adding significant user complexity; despite taking >10 minutes per image, it failed to produce meaningful results on most of the easy-category examples. Among the alternatives, VTracer yielded the most consistent and usable outlines. Although its outputs contain visible artifacts compared to Image Trace (Figure 2, top row), the overall structure is well preserved.

VTracer as a Drop-In Replacement. We additionally evaluated using VTracer in place of Image Trace in the full preprocessing pipeline. As shown in the bottom row of Figure 2, despite the noisier outline input, the final SVG quality remains comparable. Quantitatively, both vectorizers achieve a DINO score of approximately 0.98 on SVGenius, demonstrating that our pipeline is robust to the choice of vectorizer and that VTracer is a viable open-source alternative.



(a) Input (b) Potrace (c) VTracer (d) Image Trace

(e) Ours (VTracer input) (f) Ours (Image Trace input)

Figure 2. Vectorizer comparison and pipeline robustness on a sample input.

1.3. Intricate Reconstruction

Figure 3 highlights the expressive power of the rounded polygon representation on a highly intricate vector graphic: a public-domain wolf-head silhouette[1] with roughly 300 control points and dense local structure. Even at this level of geometric complexity, the reconstruction remains closely aligned with the original SVG while requiring substantially fewer tokens than raw SVG commands. This example shows that the proposed representation is not restricted to simple icon-like geometry, but can also support detailed and visually complex designs in a compact form.

[1] https://freesvg.org/wolf-head-silhouette

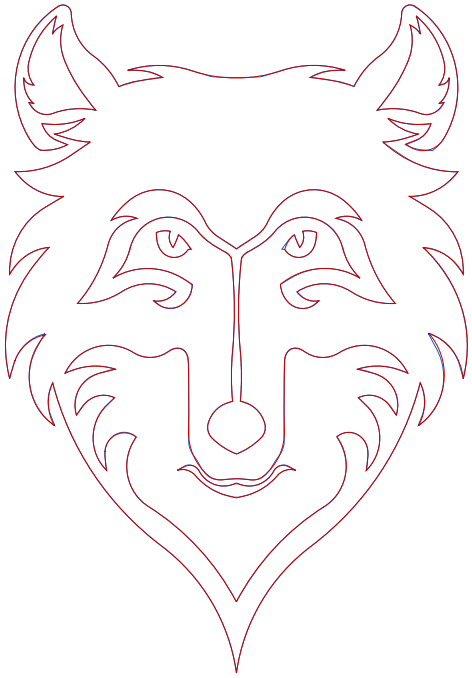
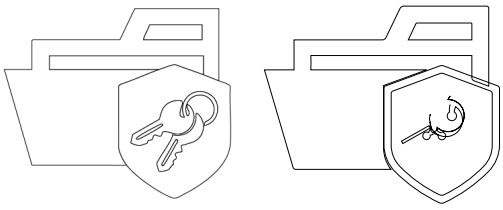


Figure 3. Intricate reconstruction example. Blue: original SVG; red: our reconstruction.

1.4. Failure Case

Figure 4 shows a representative failure mode. When many paths are concentrated in a very small spatial region, the model struggles to resolve fine-grained local topology and tends to merge or misplace individual primitives. We observe that increasing the input raster resolution alleviates this issue and produces noticeably better reconstructions in such cases.



(a) Vectorized input (b) Model output

Figure 4. Failure case with dense local path structure concentrated in a small region.

1.5. Additional Metrics and Analysis

To further substantiate our claims, we report complementary evaluations beyond those in the main paper: results without test-time scaling, a geometry-level metric (Chamfer distance), a qualitative analysis of reconstruction behavior on easy-category samples, and an inference-time comparison.

1.5.1. Evaluation Without Test-Time Scaling

Table 1 reports performance with a single stochastic sample (N=1), *i.e.* without best-of-N selection. Our method achieves the best scores across all four metrics, maintaining a clear lead even without test-time scaling.

Table 1. Performance without test-time scaling (N=1) on SVGenius (averaged across Easy/Medium/Hard).

Metric	GPT-4o	Gemini	OmniSVG	StarVector	Ours
SSIM ↑	0.590	0.540	0.679	0.724	**0.815**
LPIPS ↓	0.274	0.330	0.232	0.226	**0.136**
MSE ↓	0.094	0.124	0.066	0.065	**0.033**
DINO ↑	0.929	0.886	0.885	0.862	**0.940**

1.5.2. Chamfer Distance

Beyond raster-space evaluation, we additionally report a metric that operates directly in the vector domain and is better suited to assessing geometric reconstruction quality. Table 2 reports Chamfer distance between predicted and ground-truth control points, a metric better aligned with the geometric nature of SVG outputs. Our method achieves the lowest Chamfer distance on Medium and Hard splits, and matches OmniSVG on Easy, reinforcing the strong reconstruction quality observed with raster-space metrics.

Table 2. Chamfer distance (↓) between predicted and ground-truth control points.

Difficulty	GPT-4o	Gemini	OmniSVG	StarVector	Ours
Easy	0.067	0.080	**0.024**	0.055	**0.024**
Medium	0.080	0.093	0.063	0.108	**0.054**
Hard	0.081	0.086	0.073	0.133	**0.071**

1.5.3. Reconstruction Behavior on Easy Samples

As noted in the ablation study, in the main paper, the raw SVG-command representation attains slightly stronger pixel-level metrics on the Easy split while our rounded polygon representation leads in semantic similarity (DINO). Figure 5 illustrates why: our representation regularizes minor local irregularities, so the predicted paths (red) may exhibit small geometric deviations from the ground truth (black). On easy samples, which contain only a few primitives, each such deviation has a proportionally larger effect on pixel-level metrics. However, DINO—which captures higher-level shape semantics—remains equal or better, indicating that the overall structure is faithfully preserved. On Medium and Hard samples this effect is less pronounced, as the dominant source of error shifts to broader geometric misalignment where our representation's stability becomes the decisive advantage.

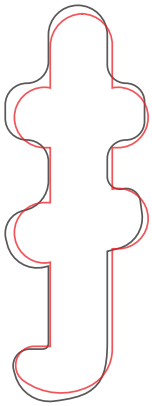


Figure 5. Easy-category reconstruction: our output (red) versus ground truth (black). The rounded polygon representation smooths minor irregularities while preserving overall shape fidelity.

1.5.4. Inference Time

On an A100, our method generates one SVG in 3–4 s on easy-category samples, compared with 8–16 s for StarVector and 4–6 s for OmniSVG. The compact 1B architecture (vs. 8B for StarVector and 3B for OmniSVG) enables both faster generation and lower memory consumption. Classical image vectorization adds less than 1 s at 1024×1024 resolution, and batched DINO scoring for test-time scaling adds < 0.25 s. The same vectorized outline can be reused across multiple stochastic samples, further amortizing preprocessing cost.

1.6. Extended Benchmark Evaluation

1.6.1. Setup

We evaluate on ten benchmarks from three recent works: eight from StarVector [3] (**SVG-Stack**, **SVG-Stack-Simple**, **SVG-Fonts**, **SVG-Fonts-Simple**, **SVG-Icons**, **SVG-Icons-Simple**, **SVG-Emoji**, **SVG-Emoji-Simple**), as well as **MMSVG-Icon** [6] and **SVGX_core_250k** [5]. These benchmarks encompass diverse vector graphic domains including fonts, icons, emoji, and multi-colored graphics. To systematically assess performance across geometric complexity, we categorize samples into three difficulty tiers based on path primitive counts: *Easy* (< 64), *Medium* (64–128), and *Hard* (> 128).

1.6.2. Quantitative Results

Tables 3, 4, and 5 present comprehensive quantitative comparisons across all benchmarks. Our method consistently achieves state-of-the-art performance across the majority of datasets, metrics, and complexity tiers.

On non-simplified datasets (*SVG-Stack*, *SVG-Fonts*, *SVG-Emoji*, *MMSVG-Icon*, *SVGX_core_250k*), our method ranks first across all four evaluation metrics on the majority of complexity tiers. The performance advantages are particularly pronounced on higher complexity levels, where our geometry-first approach demonstrates stronger capability in reconstructing intricate path topology.

On simplified datasets, the results remain highly competitive. For *SVG-Emoji-Simple* and *SVG-Fonts-Simple*, we lead across all metrics and complexity levels. On *SVG-Icons* and *SVG-Icons-Simple*, while OmniSVG achieves competitive perceptual similarity (LPIPS) on certain subsets, our method maintains stronger structural fidelity (SSIM) and competitive semantic coherence (DINO).

2. Post-Processing: Color and Stroke Recovery

As described in the main paper, our model predicts colorless geometry from outline-based inputs. This design simplifies the learning task and improves robustness to appearance variations. Practical vectorization, however, also requires recovering colors and stroke properties from the original input image. We present a comprehensive post-processing pipeline that leverages geometric congruence between predicted and input geometries to enable efficient color recovery, followed by z-order refinement and stroke detection.

2.1. Geometric Congruence and Color Recovery

Training-Induced Geometric Alignment. A key property of our approach stems from the training data construction. During training, we ensure that the ground-truth SVG spatially aligns with the input outline raster image. As a result, this geometric alignment carries over to inference: the model learns to predict rounded polygons that, when reconstructed to SVG, produce outlines spatially congruent with the input raster's underlying structure. This property enables efficient color recovery through overlap-based voting (as described below).

Overlap-Based Voting for Color Assignment. Given predicted paths $\mathcal{P} = \{P_1, \ldots, P_K\}$ and input raster I_{src}, we recover colors through a voting mechanism. A key challenge is that paths spatially overlap in the rendered output, making it ambiguous which input pixels should determine each path's fill color. To resolve this ambiguity, we must identify the pixels that exclusively correspond to each path—avoiding contamination from overlapping neighbors.

We determine this correspondence through a mask-based subtraction approach. For each path P_i, we construct a binary mask M_i by rasterizing P_i at its spatial position within the SVG coordinate frame (at resolution r, typically 256×256). Critically, all masks $\{M_1, \ldots, M_K\}$ are rasterized in the same coordinate space, preserving their relative positions—paths that are spatially distant will have non-intersecting masks, while overlapping paths will have intersecting masks. To isolate the exclusive pixel set for P_i, we compute the set difference: subtract the union of all other paths' masks from M_i. Formally, the exclusive mask

Table 3. Quantitative comparison on non-simplified StarVector benchmarks. Best results in **bold**, second best underlined.

Method	Metric	SVG-Stack			SVG-Fonts			SVG-Icons			SVG-Emoji		
		Easy	Med	Hard	Easy	Med	Hard	Easy	Med	Hard	Easy	Med	Hard
OmniSVG	LPIPS↓	0.2297	0.2572	0.3156	0.0515	0.0920	0.1524	0.0670	0.1287	**0.1734**	0.2706	0.4266	0.4275
	SSIM↑	0.6128	0.6230	0.5787	0.8962	0.8238	0.7323	0.8507	0.7658	0.7103	0.6633	0.5304	0.4930
	DINO↑	0.8976	0.8835	0.8523	0.9743	0.9468	0.9183	0.9755	0.9473	**0.9339**	0.9023	0.8621	0.8461
	MSE↓	0.1225	0.0839	0.1081	0.0269	0.0571	0.0970	0.0346	0.0642	0.0879	0.0749	0.1100	0.1201
GPT-4o	LPIPS↓	0.2287	0.2452	0.2728	0.1579	0.1944	0.2393	0.1637	0.2069	0.2330	0.2423	0.3488	0.3891
	SSIM↑	0.5806	0.6120	0.5825	0.7513	0.7089	0.6287	0.7543	0.6898	0.6560	0.6678	0.5672	0.5253
	DINO↑	0.9541	0.9399	0.9238	0.9647	0.9326	0.9001	0.9565	0.9161	0.9111	0.9508	0.8975	0.8336
	MSE↓	0.1447	0.0942	0.1168	0.1186	0.1258	0.1650	0.0682	0.0973	0.1191	0.0719	0.1060	0.1070
Gemini-2.5-Pro	LPIPS↓	0.2537	0.2913	0.2795	0.1786	0.2077	0.2431	0.2070	0.2417	0.2586	0.2946	0.4001	0.3942
	SSIM↑	0.5370	0.5673	0.5704	0.7280	0.6950	0.6298	0.6971	0.6309	0.6136	0.6201	0.5280	0.4870
	DINO↑	0.9485	0.9328	0.9310	0.9384	0.9036	0.9086	0.9374	0.9249	0.9138	0.9385	0.9066	0.9013
	MSE↓	0.1740	0.1183	0.1109	0.1322	0.1448	0.1648	0.1063	0.1394	0.1436	0.0917	0.1174	0.1087
StarVector_8B	LPIPS↓	0.0609	0.1166	0.1933	0.0624	0.1194	0.1961	0.0764	0.1383	0.2146	0.0857	0.2683	0.3605
	SSIM↑	0.8714	0.8114	0.7046	0.8754	0.7978	0.7069	0.8532	0.7712	0.7109	0.8503	0.6588	0.5883
	DINO↑	0.9774	0.9503	0.9180	0.9756	0.9192	0.8489	0.9606	0.9261	0.8476	0.9875	0.9200	0.8411
	MSE↓	0.0275	0.0357	0.0766	0.0396	0.0694	0.1285	0.0357	0.0669	0.0961	0.0220	0.0804	0.0886
Ours	LPIPS↓	**0.0274**	**0.0577**	**0.0643**	**0.0217**	**0.0277**	**0.0555**	**0.0565**	**0.0831**	0.1845	**0.0282**	**0.0709**	**0.1517**
	SSIM↑	**0.9531**	**0.9150**	**0.9146**	**0.9651**	**0.9505**	**0.9074**	**0.9249**	**0.8683**	**0.7193**	**0.9588**	**0.8946**	**0.7971**
	DINO↑	**0.9905**	**0.9744**	**0.9791**	**0.9912**	**0.9868**	**0.9773**	**0.9814**	**0.9746**	0.9259	**0.9917**	**0.9841**	**0.9602**
	MSE↓	**0.0044**	**0.0064**	**0.0111**	**0.0075**	**0.0073**	**0.0209**	**0.0166**	**0.0323**	**0.0769**	**0.0043**	**0.0142**	**0.0288**

Table 4. Quantitative comparison on simplified StarVector benchmarks. Best results in **bold**, second best underlined.

Method	Metric	SVG-Stack-Simple			SVG-Fonts-Simple			SVG-Icons-Simple			SVG-Emoji-Simple		
		Easy	Med	Hard	Easy	Med	Hard	Easy	Med	Hard	Easy	Med	Hard
OmniSVG	LPIPS↓	0.0911	0.1712	0.2201	0.0766	0.1748	0.2240	0.0736	**0.1202**	**0.1453**	0.0922	0.2920	0.3416
	SSIM↑	0.7203	0.6078	0.5698	0.8196	0.6992	0.6530	0.8462	0.7976	0.7603	0.7451	0.4376	0.4226
	DINO↑	0.9696	0.9308	0.8901	0.9659	0.9208	0.8787	0.9714	0.9493	0.9362	0.9709	0.8761	0.8257
	MSE↓	0.0919	0.1302	0.1706	0.0437	0.0830	0.1143	0.0370	0.0578	0.0781	0.0567	0.1954	0.1867
GPT-4o	LPIPS↓	0.2049	0.2569	0.2552	0.2004	0.2465	0.2659	0.1553	0.1756	0.2009	0.2350	0.3491	0.3629
	SSIM↑	0.5978	0.5369	0.5780	0.7125	0.6365	0.6041	0.7660	0.7292	0.7025	0.5871	0.3867	0.4169
	DINO↑	0.9491	0.9294	0.8914	0.9358	0.9022	0.8921	0.9500	0.9451	0.9198	0.9122	0.8437	0.8030
	MSE↓	0.1100	0.1429	0.1334	0.0802	0.1229	0.1388	0.0631	0.0872	0.1036	0.1367	0.1893	0.1905
Gemini-2.5-Pro	LPIPS↓	0.2535	0.2781	0.2784	0.2004	0.2569	0.2682	0.2049	0.2362	0.2305	0.2754	0.3720	0.3861
	SSIM↑	0.5497	0.5072	0.5319	0.6963	0.6123	0.5987	0.7049	0.6507	0.6578	0.5322	0.3587	0.3599
	DINO↑	0.9259	0.9133	0.9066	0.9195	0.8961	0.9019	0.9308	0.9264	0.9258	0.8961	0.8800	0.8611
	MSE↓	0.1704	0.1822	0.1683	0.1217	0.1583	0.1501	0.1048	0.1351	0.1301	0.1865	0.2014	0.2338
StarVector_8B	LPIPS↓	0.0824	0.1679	0.2295	0.0610	0.1589	0.2164	**0.0686**	0.1338	0.1727	0.1094	0.2865	0.3409
	SSIM↑	0.7626	0.6384	0.6294	0.8425	0.7058	0.6547	0.8582	0.7980	0.7598	0.7436	0.4953	0.4576
	DINO↑	**0.9849**	0.9162	0.8465	0.9815	0.9234	0.8828	0.9690	0.9116	0.8767	0.9620	0.8250	0.7803
	MSE↓	0.0541	0.1038	0.1052	0.0349	0.0747	0.1075	0.0331	0.0631	0.0823	0.0561	0.1132	0.1441
Ours	LPIPS↓	**0.0390**	**0.0425**	**0.0574**	**0.0339**	**0.0787**	**0.1674**	0.0798	0.1209	0.1697	**0.0623**	**0.1008**	**0.2740**
	SSIM↑	**0.9460**	**0.9361**	**0.9246**	**0.9435**	**0.8758**	**0.7492**	**0.8995**	**0.8188**	**0.7606**	**0.9151**	**0.8465**	**0.6146**
	DINO↑	0.9823	**0.9832**	**0.9653**	**0.9903**	**0.9729**	**0.9459**	**0.9722**	**0.9559**	**0.9400**	**0.9772**	**0.9593**	**0.8718**
	MSE↓	**0.0086**	**0.0103**	**0.0155**	**0.0111**	**0.0271**	**0.0545**	**0.0210**	**0.0478**	**0.0631**	**0.0129**	**0.0253**	**0.0758**

is $M_i^{\text{excl}} = M_i \setminus \bigcup_{j \neq i} M_j$. The white pixels in M_i^{excl} constitute the regions that uniquely belong to P_i without being occluded by any other path. We then sample colors from I_{src} at these exclusive pixel locations and assign the median (per RGB channel) as the fill color for P_i.

However, when paths overlap heavily, some paths may have empty exclusive masks ($M_i^{\text{excl}} = \emptyset$). In practice, we address this using an iterative strategy: after each pass, paths that have been successfully assigned colors are removed from subsequent iterations. This progressively reveals previously occluded paths, allowing them to be colored in later passes. The process continues until all paths have been assigned colors or no further progress can be made. A possible extension would be to analyze the color

Table 5. Quantitative comparison on additional benchmarks. Best results in **bold**, second best underlined.

Method	Metric	MMSVG-Icon			SVGX_core_250k		
		Easy	Med	Hard	Easy	Med	Hard
OmniSVG	LPIPS↓	0.0758	0.1210	0.1357	0.0651	0.1576	0.2513
	SSIM↑	0.8230	0.7581	0.7337	0.8446	0.7091	0.5978
	DINO↑	0.9869	0.9711	0.9665	0.9871	0.9552	0.9242
	MSE↓	0.0277	0.0439	0.0588	0.0300	0.0690	0.0883
GPT-4o	LPIPS↓	0.2156	0.2602	0.2831	0.2055	0.2649	0.3237
	SSIM↑	0.6081	0.5579	0.5276	0.6299	0.5416	0.5193
	DINO↑	0.9737	0.9594	0.9403	0.9680	0.9434	0.8726
	MSE↓	0.1209	0.1271	0.1496	0.1408	0.1473	0.1259
Gemini-2.5-Pro	LPIPS↓	0.2550	0.3220	0.3388	0.2322	0.3141	0.3438
	SSIM↑	0.5712	0.4945	0.4813	0.6004	0.5043	0.4711
	DINO↑	0.9466	0.9220	0.9171	0.9441	0.9261	0.9074
	MSE↓	0.1502	0.1649	0.1723	0.1647	0.1757	0.1540
StarVector_8B	LPIPS↓	0.0451	0.0964	0.1431	0.0409	0.1053	0.2360
	SSIM↑	0.9022	0.8091	0.7292	0.9101	0.8086	0.6559
	DINO↑	0.9924	0.9783	0.9551	0.9910	0.9625	0.9058
	MSE↓	0.0151	0.0433	0.0627	0.0212	0.0482	0.0837
Ours	LPIPS↓	**0.0122**	**0.0179**	**0.0324**	**0.0168**	**0.0332**	**0.0677**
	SSIM↑	**0.9736**	**0.9675**	**0.9452**	**0.9661**	**0.9484**	**0.8953**
	DINO↑	**0.9974**	**0.9961**	**0.9924**	**0.9938**	**0.9889**	**0.9809**
	MSE↓	**0.0024**	**0.0033**	**0.0070**	**0.0047**	**0.0066**	**0.0146**

distribution of pixels corresponding to each path in the raster input: if multiple peaks are present and the dominant color has already been assigned to a neighboring path in earlier iterations, selecting the next most significant unassigned peak may yield better results.

2.2. Z-Order Optimization

After the iterative color extraction process assigns colors to all paths using the initial ordering π_0 (the order in which polygons appear in the predicted representation), we perform z-order optimization to further improve reconstruction quality. The optimization searches for alternative orderings that minimize rendering error against I_{src}. The key steps are: (1) construct an overlap graph where paths with intersecting binary masks are connected by edges, (2) decompose this graph into independent connected components, (3) generate valid topological orderings within each component while respecting subset constraints, (4) prune the search space based on valid orderings, and (5) evaluate orderings via MSE comparison.

2.3. Z-Order Optimization Details

Problem Formulation. Given initially colored paths $(\mathcal{P}, \mathcal{C}_0)$ in the initial ordering π_0, we seek the optimal ordering that minimizes rendering error:

$$\pi^* = \arg\min_{\pi} \text{MSE}(\text{Render}(\mathcal{P}, \mathcal{C}_0, \pi), I_{\text{src}}) \quad (1)$$

The optimization returns the single best ordering π^* that achieves minimum MSE.

Connected Component Decomposition. We construct an overlap graph $G = (V, E)$ where each vertex represents a path and edges connect overlapping paths. Finding connected components via depth-first search decomposes the problem: paths in separate components can be optimized independently. This reduces complexity from $O(K!)$ to $O(\prod_j n_j!)$ where n_j is the size of component j. For medium-complexity artworks, we found that most components have $n_j \leq 5$, making exhaustive enumeration tractable.

Topological Ordering with Pruning. Within each component, subset constraints form a directed acyclic graph (DAG): if $P_i \subset P_j$, then P_j must render before P_i (to avoid incorrect occlusion). We generate orderings via backtracking with topological constraints. Two key optimizations: (1) *Independent pairs*: paths that don't overlap have commutative order—we select canonical representatives during generation to avoid redundant branches; (2) *Early termination*: stop at 120 evaluations and use area-based fallback (sort by area descending).

Evaluation and Selection. We optimize each component independently. For each component, we generate candidate orderings via the procedure described above (up to 120 evaluations per component). To evaluate a component's orderings in isolation, we mask out all other components from I_{src} and compute MSE only over the component's spatial region. This yields the optimal internal ordering for that

component. Since paths in different components do not overlap, their relative inter-component order does not affect rendering. The final global ordering π^* preserves the optimized intra-component orderings while allowing arbitrary arrangement of components relative to each other.

2.4. Source-Guided Stroke Detection

Motivation. Many vector graphics include strokes (outlines) in addition to fills. We employ an analytical method to determine stroke properties by directly measuring them from the input raster image.

Detection and Validation. For each path, we sample points along its geometry and compute the perpendicular (normal) direction at each point. We then traverse perpendicular to the path direction, stepping pixel-by-pixel until the color changes significantly. This directly measures the half-width of the stroke, and traversing in both normal directions ($+\mathbf{n}$ and $-\mathbf{n}$) yields the total stroke width. Individual measurements may vary due to anti-aliasing, path curvature, or sampling artifacts. One approach to handle this variability is to cluster width measurements and color measurements independently, then derive consensus values by selecting the median of the largest width cluster and the mode of the largest color cluster, which can provide robustness to outliers. To avoid false positives (e.g., detecting fills as strokes), we validate each detected stroke by rendering the SVG with the proposed stroke and computing pixel-level MSE against the source image. Strokes are accepted only if they improve MSE by at least $\epsilon = 0.0002$.

Pipeline Ordering. We perform fill color extraction and z-order optimization before stroke detection because, in most vector graphics, the majority of pixels correspond to path fills rather than strokes. By first optimizing fill colors and path ordering, which account for most of the reconstruction error, we substantially reduce MSE before adding strokes as a final refinement. In practice, this staged design is also more computationally efficient.

Practical Performance and Future Work. In our evaluation, this pipeline produces correct results for the vast majority of test cases while maintaining efficient computational cost ($\sim$200 $-$ 300ms per image on a single CPU core). The geometric congruence from training combined with iterative visibility-aware color extraction handles most occlusion scenarios effectively. However, these methods can fail on very complex SVG structures and cannot address all SVG properties (e.g., gradients, filters, transparency effects). Despite these limitations, we found them to work well in practice. Further improvement in handling complex structures and extending support for additional SVG properties remains an important direction for future work.

References

[1] 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 conference on computer vision and pattern recognition*, 2022. 1

[2] Sanford Pun and Chris Tsang. VTracer: Raster to Vector Graphics Converter, 2020. Vision Cortex documentation. 1

[3] Juan A Rodriguez et al. Starvector: Generating scalable vector graphics code from images. *arXiv preprint arXiv:2312.11556*, 2025. 3

[4] Peter Selinger. Potrace: a polygon-based tracing algorithm. http://potrace.sourceforge.net/, 2003. 1

[5] Ximing Xing, Juncheng Hu, Guotao Liang, Jing Zhang, Dong Xu, and Qian Yu. Empowering llms to understand and generate complex vector graphics. *arXiv preprint arXiv:2412.11102*, 2024. 3

[6] Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Jiaxu Zhang, Liao Wang, Gang Yu, Xinjun Ma, and Yu-Gang Jiang. Omnisvg: A unified scalable vector graphics generation model. *arXiv preprint arxiv:2504.06263*, 2025. 3