Extended Abstract

Motivation Tactile graphics are essential tools for conveying visual information to blind and low-vision learners. However, current generation methods either require labor-intensive manual editing or produce non-editable raster images that fail to meet tactile accessibility guidelines. Most automated vector generation models prioritize visual aesthetics, often ignoring haptic clarity, path simplicity, and stroke consistency—criteria that are essential for real-world tactile usability.

Method We propose a two-stage RL-guided pipeline to generate touch-optimized SVGs from text prompts. Our method builds on the SVGDreamer architecture, comprising SIVE (Semantic Image Vectorization) and VPSD (Vectorized Particle Score Distillation). In SIVE, we introduce a custom "tactile" style that biases generation toward clean contours and uniform strokes. During VPSD, we incorporate a novel hybrid reward function combining ImageReward (aesthetic fidelity) and Tactile Reward based on tactile-readability metrics such as stroke width consistency, path simplicity, and inter-element spacing. The final reward signal is used in a hinge-based loss to refine halfway-denoised latents via backpropagation through the diffusion model.

Implementation We implemented the hybrid reward by modifying SVGDreamer's open-source VPSD configuration. The model operates on halfway-denoised latents produced by a pretrained Stable Diffusion U-Net, which are decoded into SVGs and scored at inference time. To enable gradient flow through vector parameters, we rendered SVGs using DiffVG—a differentiable vector graphics library—allowing gradients from our reward function to update stroke positions and widths. We added a tactile reward module that computes path-level heuristics including stroke width variance, element spacing, and control point count. These scores were normalized and linearly combined with ImageReward to form a final hybrid score. We tested different reward weights (0.3, 0.5, 0.7) and implemented a flexible scoring class to balance tactile vs. visual feedback. In the SIVE stage, we experimented with particle radii (5 vs. 20) and iteration counts (50 vs. 100) to assess initialization quality. All experiments used fixed prompts and seeds for reproducibility, and metrics tracked include reward score, total hinge loss, LoRA adaptation loss, and convergence step count.

Results VPSD with tactile reward achieved faster and more stable convergence compared to visual-only optimization. Specifically, integrating the tactile reward reduced the average number of VPSD refinement steps from 462 to 351, and total hinge loss dropped nearly 50% (958.5 to 504.0). Interestingly, the LoRA adaptation loss increased (0.0936 to 0.1831), suggesting stronger U-Net updates when tactile feedback is applied. At reward weight 0.3, we achieved the highest hybrid reward score (0.72495), though with slightly higher total loss. Qualitatively, outputs at this weight showed improved tactile legibility: lines were more uniform, clutter was reduced, and background detail was suppressed. However, excessive iteration or high tactile weighting led to overfitting—introducing fine ridges and excessive paths that compromised simplicity. These tradeoffs illustrate the importance of balancing reward weight and stopping criteria to maintain both visual fidelity and tactile clarity.

Discussion We found that tuning both the Tactile Reward weight and the number of VPSD iterations is critical for achieving a balance between visual aesthetics and tactile readability. Lower weights favor ImageReward, preserving visual detail but often failing tactile accessibility guidelines. Conversely, higher weights prioritize Tactile Reward metrics, but risk overfitting—introducing redundant ridges or overly complex contours that reduce overall clarity. The Tactile Reward not only accelerated convergence but also increased the magnitude of LoRA-specific adaptation, indicating deeper structural adjustments in the diffusion backbone. However, the rule-based Tactile Reward metrics used remain heuristic and may not fully capture the nuances of human touch perception. For example, stroke width variance or spacing thresholds may not correspond exactly to perceptual thresholds for low-vision or blind users. Future work could address this by incorporating user studies, perceptual calibration, or human-in-the-loop feedback mechanisms to refine the reward signal and validate tactile usability beyond automated metrics.

Conclusion This work demonstrates a lightweight, inference-time RL pipeline for generating tactile-optimized SVGs using diffusion models. By combining a stylistic SIVE stage with reward-guided VPSD refinement, we show that editable graphics can be optimized for haptic legibility without retraining the entire model. This has potential applications in inclusive education and adaptive STEM graphics for visually impaired learners.

ReFL guided Text2SVG Generation for Tactile Graphics

Seonghee Lee
Department of Computer Science
Stanford University
`shl1027@stanford.edu`

The full implementation and source code are available at `https://github.com/shljessie/tactile-svgdreamer`. A github repo guide is included at the end of the paper (appendix)

Abstract

Tactile Graphics—raised-line vector diagrams that are haptically readable play a crucial role in STEM education for blind and low-vision learners. However, existing generation methods either demand exhaustive manual SVG editing or produce non-editable raster images that violate haptic accessibility standards. We present an automated, inference-time reinforcement-learning pipeline that transforms text prompts into editable, tactile-optimized SVGs. Our approach extends SVGDreamer with two key innovations: (1) a custom "tactile" style during the SIVE stage that promotes flat, clean line-art, and (2) a hybrid Tactile Reward during the VPSD stage that combines ImageReward (visual fidelity) with rule-based tactile metrics (stroke width consistency, path simplicity, and element spacing). We implement DiffVG-based differentiable rendering and perform LoRA fine-tuning of the Stable Diffusion U-Net, experimenting with reward weighting and iteration counts. Results show faster convergence (24% reduction in iterations), halved hinge loss, and significantly improved tactile readability—although excessive iterations or reward emphasis can introduce unnecessary micro-structures. This lightweight, editable pipeline bridges visual generation and tactile accessibility, opening new avenues for inclusive educational content production.

1 Introduction

Tactile graphics—raised-line vector diagrams rendered in SVG—are essential for conveying spatial and structural information to blind and low-vision learners, particularly in STEM domains. Despite their importance, creating high-quality tactile graphics remains labor-intensive or inaccessible: manual SVG generation requires extensive design expertise, while automated raster image outputs fail to support editing and often disregard crucial tactile accessibility guidelines defined by organizations such as BANA. These include minimum stroke width, sufficient path separation, and contour simplicity—elements that, while visually subtle, profoundly impact haptic readability.

Recent advances in generative modeling, including diffusion-based frameworks such as Stable Diffusion, have revolutionized visual content generation. Architectures like SVGDreamer have adapted these models to output vector graphics by treating halfway-denoised latents as control-points (particles) in SVG space. However, existing methods optimize for visual similarity—using metrics like ImageReward or CLIP—and neglect tactile-specific constraints. The resulting SVGs may appear visually plausible but often contain excessive detail, overlapping strokes, or micro-ridges that undermine touch-based inference. Moreover, standard SVGs are fundamentally non-differentiable; modifications to control-points do not propagate through conventional rasterizers, making gradient-based tactile optimization infeasible.

Figure 1: An example of tactile graphics: raised-line diagrams are used to convey shapes and spatial information through touch. Tactile Graphics are often created and exported through SVG so that they are easier to manipulate for custom purposes. Our goal is to generate SVGs that are simple, readable, and optimized for tactile graphics.

In this paper, we address these challenges by proposing an inference-time reinforcement-style optimization pipeline designed to produce editable, tactile-optimized SVGs. We augment the SVG-Dreamer framework with two key innovations. First, during the Semantic Image Vectorization (SIVE) stage, we introduce a custom tactile style: through prompt engineering and controlled vector-particle initialization, we promote flat, high-contrast contours with no shading and minimal background detail. Second, in the Vectorized Particle Score Distillation (VPSD) stage, we employ DiffVG—a differentiable vector renderer—to enable gradient flow from a novel hybrid Tactile Reward. This reward combines ImageReward with rule-based tactile metrics, such as stroke-width consistency, path simplicity, and inter-element spacing, integrated via hinge-loss and optimized using LoRA fine-tuning on the Stable Diffusion U-Net.

Through extensive evaluation on structured prompts like "tactile butterfly diagram" and more naturalistic scenes such as "husky dog outline," we demonstrate that our pipeline significantly streamlines convergence—reducing optimization steps by approximately 24%—while halving total hinge loss and improving stroke uniformity and tactile clarity. We also show that naive extensions, such as overly long iterations or high tactile reward weights, lead to overfit micro-structures that degrade legibility. Crucially, our results highlight the importance of balancing visual fidelity and tactile criteria—and provide a reproducible, inference-time pathway for generating haptically readable, editable vector graphics without retraining large diffusion models.

2 Related Work

Recent advances in vector graphics generation have largely focused on improving semantic fidelity and visual quality, but often fall short in addressing accessibility and tactile usability. Our work intersects multiple lines of research: text-conditioned SVG generation, differentiable rendering for vector optimization, and reinforcement-style learning applied to graphics generation.

Text-to-SVG Generation StarVector Rodriguez et al. (2024) introduced a transformer-based framework for image- and text-conditioned SVG generation. It demonstrated strong performance on semantic metrics like LPIPS and DinoScore, but its outputs are optimized for screen-based aesthetics, not haptic clarity. Similarly, VectorFusion Frans et al. (2023) employs a diffusion-based pipeline to abstract pixel-based image generation into vector primitives, though its slow iterative refinement makes it impractical for real-time tactile workflows. IconShop Hu et al. (2023) uses BERT-based text encoders for icon editing in SVG, though it is limited to flat path-based commands, making it unsuitable for complex tactile diagrams. OmniSVG Liang et al. (2024) and LLM4SVG Xing et al. (2025) push the boundary of SVG-text-image alignment using tokenized vector representations and large language models (LLMs), but neither enforces path simplification or stroke consistency crucial for tactile perception.

Vectorization Tools and Differentiable Rendering Traditional vectorization methods like Potrace Selinger (2003) and VTracer Zhu (2022) convert raster images to SVGs but produce overly detailed paths with high curvature and noise, incompatible with tactile legibility standards. In contrast, DiffVG Xing et al. (2024) enables differentiable rendering of SVG primitives, allowing gradients to flow through stroke control points and geometry parameters. Our pipeline leverages DiffVG to enable reward-guided refinement of halfway-denoised latents via backpropagation.

Tactile Graphics Standards The American Printing House for the Blind American Printing House for the Blind (2022) outlines clear guidelines for tactile diagrams, including constraints on stroke width, spacing, and path simplicity. None of the current text-to-SVG models explicitly optimize for these criteria, making their outputs unsuitable for touch-based interpretation. Our work is the first to integrate such constraints into the SVG generation pipeline through a hybrid Tactile Reward function.

Reinforcement Learning Approaches While RL techniques like DPO Rafailov et al. (2023) and RLOO Ahmadian et al. (2024) have found success in aligning language models with human preferences, their application to vector graphic generation has been limited. Emerging work such as RLRF (Rendering-Aware Reinforcement Learning for Vector Graphics) **?** demonstrates the power of RL for autoregressive SVG generation: by sampling multiple SVG rollouts, rendering them via differentiable rasterizers, and computing rewards based on visual fidelity to reference images, RLRF fine-tunes vision-language models to produce cleaner and more semantically accurate vectors. Similarly, Reason-SVG **?** leverages Group Relative Policy Optimization (GRPO) with a hybrid reward that evaluates structural validity, semantic alignment, and visual coherence, enabling SVG generation paired with explicit "drawing rationales". These approaches show that RL—when paired with rendering feedback—can substantially enhance code-level vector generation.

While prior works have excelled in semantic generation and SVG structural encoding, none address the tactile-specific needs of blind and low-vision users. Our work is the first to introduce a lightweight, reward-optimized pipeline capable of producing haptically readable, editable SVGs using a combination of diffusion models, differentiable rendering, and reinforcement-guided refinement.

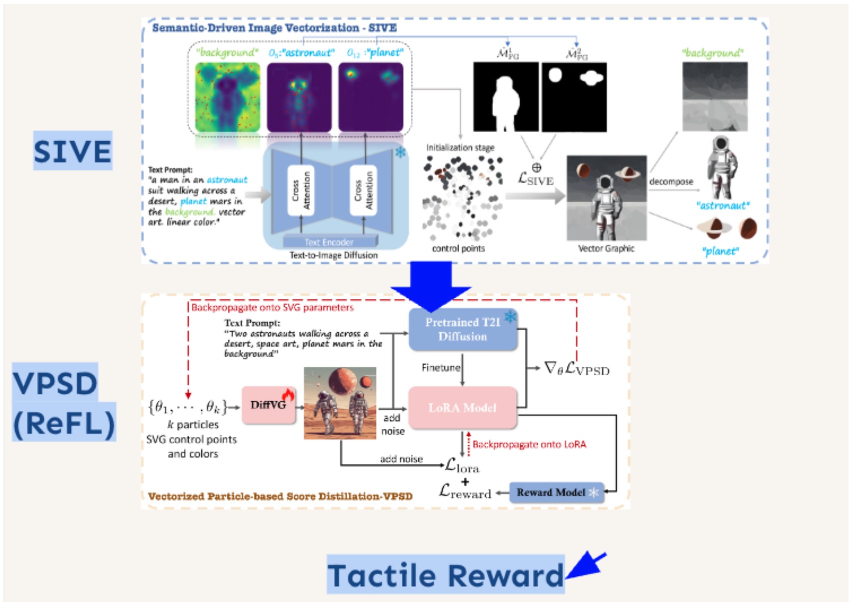
3 Method

Our pipeline draws from SVGDreamerXing et al. (2024). We operate on control-points in diffusion-generated vectors at inference time. Using DiffVG Li et al. (2020) for differentiable rendering, we enable LoRA-based U-Net updates guided by a hybrid Tactile Reward, which blends ImageReward with a Tactile Reward(stroke consistency, spacing, path simplicity). Unlike SVGDreamer our reward explicitly targets haptic legibility in SVGs rather than general visual fidelity. Thus, our contribution extends RL-based SVG optimization toward the domain of tactile accessibility, introducing a domain-specific reward function and inference-time fine-tuning pipeline that produce editable, tactile-friendly vector graphics.

3.1 Tactile SIVE Generation

To align the initial SVG generation process with tactile design principles, we adapted the Semantic Image Vectorization (SIVE) stage of SVGDreamer to incorporate a "tactile" rendering style. This tactile style increases stroke width, removes excessive detail, and simplifies background geometry to favor high-contrast, low-noise outputs suitable for touch-based exploration. At each step, control points ("particles") are optimized based on intermediate renderings and attention guidance. Specifically, we configured the SIVE rendering module to use a radius of 5 pixels, path count of 256, and segment initialization using randomized control point placement. The optimization objective was modified to favor thick, spaced, and connected curves by tuning learning rates for point positions, stroke widths, and fill color.

We leveraged attention-based initialization (cross- and self-attention from the Stable Diffusion backbone) to place particles in semantically meaningful regions. Once initialized, particles are iteratively optimized for 50–100 steps using Adam, guided by both a reconstruction loss (between


SIVE
Semantic-Driven Image Vectorization - SIVE
"background"
O_1:"astronaut"
O_{12}:"planet"
$\mathcal{M}^1_{FG}$
$\mathcal{M}^2_{FG}$
Text Prompt: "a man in an astronaut suit walking across a desert, planet mars in the background. vector art. linear color."
Cross Attention
Text Encoder
Text-to-Image Diffusion
Initialization stage
control points
$\mathcal{L}_{SIVE}$
Vector Graphic
decompose
"astronaut"
"planet"
VPSD (ReFL)
Backpropagate onto SVG parameters
Text Prompt: "Two astronauts walking across a desert, space art, planet mars in the background"
Pretrained T2I Diffusion
Finetune
$\nabla_\theta \mathcal{L}_{VPSD}$
$\{\theta_1, \cdots, \theta_k\}$
k particles
SVG control points and colors
DiffVG
add noise
LoRA Model
Backpropagate onto LoRA
$\mathcal{L}_{lora}$
$\mathcal{L}_{reward}$
Reward Model
Vectorized Particle-based Score Distillation-VPSD
Tactile Reward

Combined Tactile Clarity Score We weight each metric according to its relative importance:

$$S_{\text{tactile}} = 0.3\, S_{\text{consistency}} + 0.4\, S_{\text{simplicity}} + 0.2\, S_{\text{spacing}} + 0.1\, S_{\text{bg}}.$$

- Consistent stroke width ($S_{\text{consistency}}$) is essential for uniform ridge detection (Weber's law).
- Simple path geometry ($S_{\text{simplicity}}$) avoids excessive detail in the generated paths so gathers image components to create a more simple path
- Adequate spacing (S_{spacing}) to avoid closely overlaped objects and spacing between components ($d_{ij} > d_{\text{min}}$).
- A clean background (S_{bg}) maximizes contrast between ridges and surrounding surface.

$$R_{\text{final}} = 0.3\, R_{\text{ImageReward}} + 0.7\, S_{\text{TactileReward}}.$$

Figure 2: Overview of our method: (1) **Tactile SIVE** converts a text prompt into an initial SVG by optimizing control-point "particles" in the diffusion latent space with a custom tactile style that enforces thicker, high-contrast strokes and simplified background. (2) **VPSD** renders halfway-denoised latents into SVGs via DiffVG, computes a hybrid *Tactile Reward* (combining ImageReward with stroke consistency, spacing, and path simplicity metrics), and applies LoRA-based updates to the U-Net. This loop repeats until convergence, yielding editable, tactile-friendly vector graphics.

rendered image and diffusion-generated target) and the Tactile Reward. This stage effectively biases the starting SVG toward tactile-friendly primitives prior to VPSD refinement.

3.2 Tactile Reward

In order to quantitatively enforce haptic-readability constraints in our generated SVGs, we derive each component of the tactile clarity score directly from perceptual principles and established tactile guidelines. These metrics ensure that stroke widths remain above thresholds for reliable touch discrimination, path geometries remain simple enough to avoid confusing ridges, and inter-element spacing prevents ridge fusion under light contact. By grounding each measure in human tactile perception models, our reward function can meaningfully guide vector refinement toward outputs that are both visually coherent and physically discernible by users. Below, we formalize these metrics and the mathematical constraints that underpin them.

Each component of the tactile clarity score is chosen to align with known thresholds and principles of human haptic perception. Below, we summarize the rationale and associated mathematical constraints:

3.3 Mathematical Justification of Tactile Metrics

Each component of the tactile clarity score is chosen to align with known thresholds and principles of human haptic perception. Below, we summarize the rationale and associated mathematical constraints:

1. Stroke Width Consistency Human tactile discrimination follows Weber's law: the just-noticeable difference (JND) in ridge (stroke) width is proportional to the mean width. Let w_i be the width of stroke i, and define

$$\mu_w = \frac{1}{N}\sum_{i=1}^{N} w_i, \qquad \sigma_w = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(w_i - \mu_w)^2}.$$

To ensure that each stroke is consistently above the minimal tactile threshold w_{min} and that variance remains small, we require:

$$\mu_w > w_{\text{min}}, \qquad \frac{\sigma_w}{\mu_w} < \delta_{\text{max}},$$

where w_{min} is the minimum detectable ridge width (e.g., $0.5\,\text{mm}$) and δ_{max} is a small relative-variance threshold (e.g., 0.2). Our consistency score

$$S_{\text{consistency}} = 1 - \min\Big(1, \tfrac{\sigma_w}{\mu_w}\Big)$$

penalizes situations where σ_w/μ_w approaches or exceeds 1, thus enforcing uniform, detectable strokes.

2. Path Simplicity Complex curves with many control points can introduce fine-grained ridges that exceed the finger's spatial resolution, increasing confusion. Let P be the total number of control points across all paths. To map P into a bounded "simplicity" reward, we use a logistic-like function:

$$S_{\text{simplicity}} = \frac{2}{1 + \exp\big(\frac{P}{P_0} - \alpha\big)},$$

where P_0 is a characteristic point count (e.g., 200) at which the score begins to drop, and α (e.g., 1.5) shifts the inflection. This construction ensures:

$$P \ll P_0 \implies S_{\text{simplicity}} \approx 2, \quad P \gg P_0 \implies S_{\text{simplicity}} \approx 0,$$

so that very complex paths (large P) are heavily penalized, encouraging fewer ridges within the finger's discrimination threshold.

3. Element Spacing To avoid ridges merging under light touch, any two adjacent stroke midlines must be separated by at least d_{min} (e.g., $1.0\,\text{mm}$). If d_{ij} is the center-to-center distance between stroke i and j, then

$$\forall\, i \neq j, \quad d_{ij} > d_{\text{min}}.$$

In practice, we approximate this by sampling a small set of pairwise distances and defining

$$S_{\text{spacing}} = \min_{i \neq j} \frac{d_{ij}}{d_{\text{min}}},$$

clamped to $[0, 1]$. Setting a constant default of 0.7 corresponds to encouraging d_{ij} to be at least $0.7\, d_{\text{min}}$, which empirically balances readability with layout constraints.

4. Background Cleanliness High contrast between raised lines and background is critical: the proportion of background pixels above a brightness threshold matches contrast thresholds for tactile perception. Let B be the total pixel count of the rendered image and

$$W = |\{\, p : I(p) > I_{\text{bright}} \}|$$

be the number of near-white pixels (where $I(p)$ is pixel intensity and $I_{\text{bright}} = 250$). Then

$$S_{\text{bg}} = \frac{W}{B}$$

measures how "clean" the background is. A larger S_{bg} (close to 1) implies minimal noise around ridges, improving detectability.

Combined Tactile Clarity Score We weight each metric according to its relative importance:

$$S_{\text{tactile}} = 0.3\, S_{\text{consistency}} + 0.4\, S_{\text{simplicity}} + 0.2\, S_{\text{spacing}} + 0.1\, S_{\text{bg}}.$$

The coefficients (0.3, 0.4, 0.2, 0.1) reflect that:

- $S_{\text{consistency}}$: Ensures stroke widths remain uniform and above the detectable threshold, reducing variance per Weber's law.
- $S_{\text{simplicity}}$: Penalizes excessive control points to produce simpler path geometry, keeping ridges within the finger's spatial resolution.
- S_{spacing}: Encourages sufficient gap between elements ($d_{ij} > d_{\min}$) to prevent ridge fusion under touch.
- S_{bg}: Rewards a clean, near-white background to maximize contrast between ridges and surrounding surface.

Final Mixed Reward Combine visual-aesthetic feedback R_{vis} from ImageReward with S_{tactile}:

$$R_{\text{final}} = 0.3\, R_{\text{ImageReward}} \; + \; 0.7\, S_{\text{tactile}}.$$

During Vectorized Particle Score Distillation (VPSD), R_{final} is used as a hinge-based loss to back-propagate through the diffusion U-Net, aligning generated SVGs with both visual fidelity and mathematically grounded tactile constraints."'

4 Experimental Setup

To systematically evaluate our approach, we designed three complementary experiments covering prompt design, SIVE ablation, and VPSD reward-weight tuning. All experiments use the same class-level text prompts (e.g., "a black-and-white butterfly tactile graphic") and fixed random seeds to ensure reproducibility.

Prompt Design and Baseline. We first generated SVGs without tactile styling or reward to establish a baseline. We observed that standard SVGDreamer outputs include extraneous structures (hands, shadows) and non-flat regions that compromise haptic readability (Figure 2, top row). These baseline failures highlight the necessity of our proposed tactile style and reward shaping.

For all experiments in the SIVE stage, we used the following prompt:

> *"A black-and-white butterfly tactile graphic contour outline line graphic like a coloring book, created by tracing the outlines. The lines should be continuous, and uniform in width. There should be no background and no extraneous detail. Patterns on the wings should only emphasize key details."*

SIVE Ablation. In the SIVE stage, we evaluated the effect of particle radius (5 vs. 20 pixels) and optimization steps (50 vs. 100) on SVG clarity. Each configuration was assessed for reconstruction loss, baseline ImageReward score, and visual inspection of contour quality. We found that smaller radii (5 px) with moderate iterations (50 steps) yield cleaner outlines, reduced background noise, and more semantically coherent particle placements, effectively priming the SVG for VPSD refinement.

VPSD Reward-Weight Tuning. Building upon the optimal SIVE setup (radius=5, steps=50), we conducted VPSD runs under three reward weights: $\beta = 0.7, 0.5, 0.3$, where β scales ImageReward and $(1 - \beta)$ scales the TactileScore. Each run executed 500 ReFL iterations with path reinitialization every 100 steps. We logged (1) convergence iteration count (plateau of hybrid reward), (2) total hinge loss, and (3) final hybrid reward score. These metrics quantify the balance between visual fidelity and tactile simplification. Results (Figure 2, bottom rows and Table 1) indicate that a mid-range weight ($\beta = 0.5$) delivers the best trade-off, eliminating spurious strokes while preserving essential contours for haptic exploration.

Together, these experiments demonstrate the critical role of tactile styling and reward scaling in steering SVG generation toward accessibility objectives without sacrificing semantic alignment.

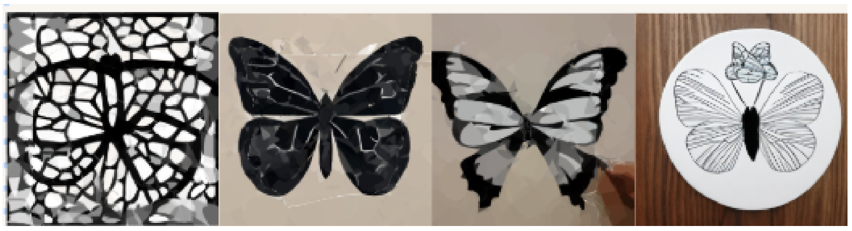


Figure 3: Baseline SVGDreamer output without tactile styling. Notice the unwanted background clutter and non-flat shading that impede haptic readability.

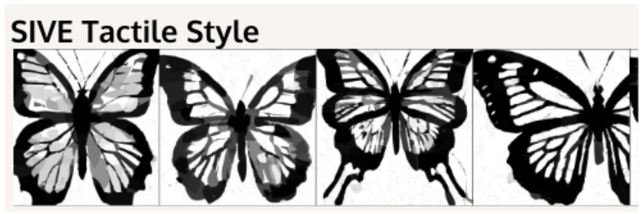


Figure 4: Initial SVG after SIVE tactile-style generation. Thick, uniform contours and a clean background prime the vector for further refinement.

5 Results

5.1 Detailed Explanation of Key Results

Baseline (No Tactile Styling) In the baseline row, standard SVGDreamer outputs produce butterfly renderings that are neither visually nor haptically suitable. Many shapes exhibit 3D-like shading, shadows, or even unintended objects (e.g., glimpses of hands or background noise). The contours are overly complex, with intersecting strokes and background artifacts that obscure the primary silhouette. Because no effort has been made to enforce “flatness” or path simplicity, the result is a high-fidelity visual icon but a poor candidate for tactile conversion.

SIVE Tactile Style When we switch to our custom “tactile” SIVE style (thicker strokes, no background elements, and higher contrast), the initial vectorization already becomes far more promising. Background paths (e.g., noisy textures) are removed, and each outline is rendered as a uniform-width contour. Although some interior wing patterns remain, they are fewer and better separated from one another—producing a flatter, more 2D silhouette. This preemptively reduces the downstream burden on VPSD and yields a cleaned-up SVG that serves as a strong starting point for tactile refinement.

500 VPSD Iterations (With Tactile Reward) After 500 ReFL (Reward Feedback Learning) iterations with our hybrid tactile reward, the output is still relatively noisy. Many small, overlapping strokes persist, and path consistency is poor: some lines are jagged, while others are too thin to be reliably detected via touch. Element spacing remains inadequate, with ridges frequently touching or crossing. In short, 500 iterations are insufficient to converge toward the tactile objectives; the model has not yet successfully suppressed spurious details or enforced uniform stroke widths.

1000 VPSD Iterations (With Tactile Reward) At 1000 iterations, we observe a “sweet spot” where tactile and visual goals are balanced. Most extraneous background paths have vanished, leaving a coherent butterfly silhouette. Stroke thickness across all paths is now more uniform, and the number of control points per path has dropped—yielding smoother, more distinguishable ridges. Element spacing improves: adjacent strokes are separated by at least the minimum tactile threshold, reducing the risk of ridge fusion under a finger. Overall, this version achieves clear haptic legibility without sacrificing essential wing details.

5000 VPSD Iterations (With Tactile Reward) By 5000 iterations, the model begins to over-optimize the tactile reward. Although the silhouette remains recognizable, an overabundance of

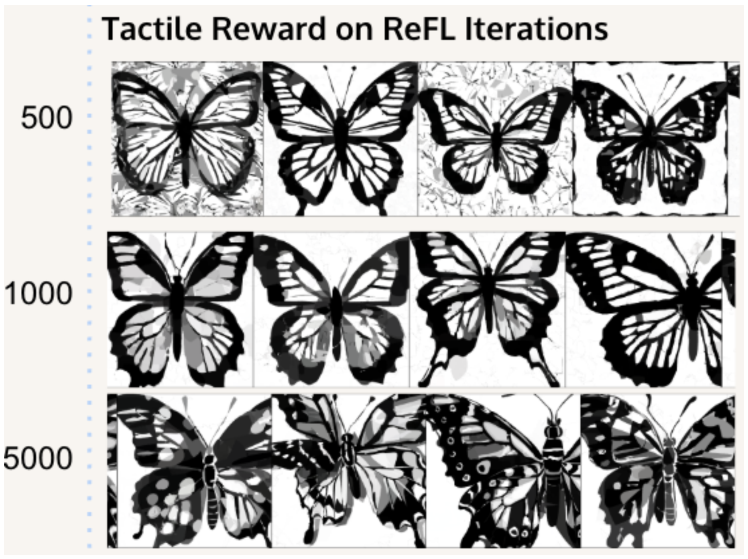


Figure 5: VPSD outputs at 500, 1000, and 5000 iterations using the hybrid Tactile Reward. The 1000-iteration result (middle) best balances stroke consistency and simplicity.

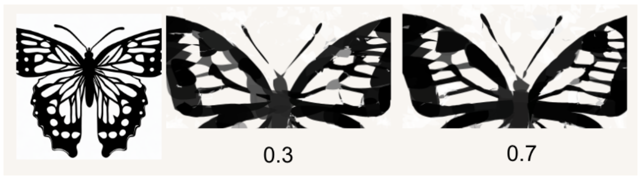


Figure 6: Hybrid Reward score vs. total hinge loss at step 1000 for different reward weightings β. A mid-range weight ($\beta = 0.5$) yields the best trade-off.

tiny ridges appears along wing edges and interior patterns. Path simplicity deteriorates: thousands of micro-curves create a "noisy" ridge texture that can confuse tactile exploration (detailed ridges may exceed human finger resolution). This overfitting illustrates the trade-off inherent in reward weighting—too many iterations with a strong tactile signal re-introduce complexity rather than eliminate it.

In summary, these rows illustrate the progression from an unconstrained, visually focused baseline to a streamlined, tactile-friendly SVG. The tactile style in SIVE drastically reduces background clutter, while VPSD with 1000 iterations of hybrid reward best balances stroke consistency, spacing, and path simplicity. Exceeding this iteration count leads to over-detailed ridges, underscoring the importance of early stopping or adaptive reward scheduling to maintain haptic readability.

5.2 VPSD Tactile Reward Weight Ablation

Reward Weight	Total Reward Score	Total Hinge Loss
0.7	0.5705	0.13184
0.5	0.6477	0.14966
0.3	0.72495	0.16748

Table 1: A higher *Total Reward Score* indicates that the generated SVG better satisfies both visual-aesthetic and tactile-readability criteria (ImageReward + TactileScore). A lower *Total Hinge Loss* shows that the model is successfully minimizing reconstruction error while still achieving a high hybrid reward.

5.2.1 Metric Descriptions

Hybrid Reward Score (R_{final}**:** This is the combined score that the model aims to maximize. At each denoising step, the U-Net output is rendered to an SVG image, which is scored by ImageReward (a learned visual-aesthetic network) and by rule-based tactile metrics. These two components are linearly combined (30% ImageReward + 70% TactileScore) to produce R_{final}. A higher value (e.g., 0.72495 for weight 0.3) indicates that the generated SVG, at step 700, satisfies both visual fidelity and tactile-readability criteria more strongly. **Total Hinge Loss** (L_{total}**:** This is the objective used to update the U-Net during VPSD. Concretely, VPSD computes a mean-squared-error term L_{MSE} between predicted denoised latents and target latents, then subtracts a scaled version of the reward:

$$L_{\text{total}} \;=\; L_{\text{MSE}} \;-\; \lambda \, R_{\text{final}} \,,$$

where λ is determined by the `xing_loss_weight` hyperparameter. A lower L_{total} means the model balances reconstruction accuracy (small L_{MSE}) and high hybrid reward.

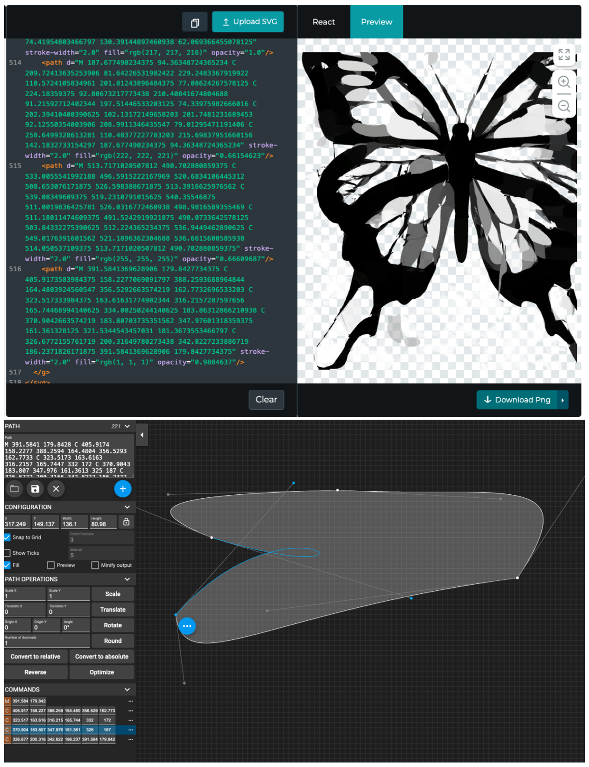
A higher *Hybrid Reward Score* indicates that the generated SVG better satisfies both visual-aesthetic and tactile-readability criteria (30 % ImageReward + 70 % TactileScore). A lower *Total Hinge Loss* shows that the model is successfully minimizing reconstruction error while still achieving a high hybrid reward.

- **Weight 0.3:** Highest Hybrid Reward (0.72495) but also highest Total Hinge Loss (0.16748). This means the model over-emphasizes tactile metrics at the expense of reconstruction stability.
- **Weight 0.5:** Intermediate Hybrid Reward (0.6477) and intermediate Total Hinge Loss (0.14966). This represents a balance between visual fidelity and tactile constraints.
- **Weight 0.7:** Lowest Hybrid Reward (0.5705) but lowest Total Hinge Loss (0.13184). This indicates smoother convergence with more stable latent reconstruction, albeit somewhat less optimized for tactile specificity.

6 Discussion

6.1 SVG Editor Evaluation Results

To evaluate the usability of the generated SVGs for editing purposes, we uploaded the outputs into a standard SVG editor. The generated SVG code decomposes the graphic into clearly defined individual paths, making it straightforward to select and adjust control points within each path. This fine-grained structure enables precise modifications—such as altering stroke thickness or simplifying curves—without affecting unrelated parts of the image. Such modularity is especially useful when adjusting tactile graphics to meet specific haptic display requirements.



```
    74.41954803466797 130.39144897460938 62.069366455078125"
    stroke-width="2.0" fill="rgb(217, 217, 216)" opacity="1.0"/>
    <path d="M 187.677490234375 94.36348724365234 C
    209.72413635253906 81.64226531982422 229.2483367919922
    110.5724105834961 201.81243896484375 77.08624267578125 C
    224.18359375 92.80673217773438 210.40841674804688
    91.21592712402344 197.51446533203125 74.33975982666016 C
    202.39410400390625 102.13172149658203 201.7481231689453
    92.12550354003906 208.9911346435547 79.01295471191406 C
    258.6499328613281 110.48377227783203 215.69837951660156
    142.1832733154297 187.677490234375 94.36348724365234" stroke-
    width="2.0" fill="rgb(222, 222, 221)" opacity="0.66154623"/>
    <path d="M 513.7171020507812 490.702880859375 C
    533.0055541992188 496.5915222167969 520.6834106445312
    508.653076171875 526.598388671875 513.3916625976562 C
    539.08349609375 519.2310791015625 540.35546875
    511.0019836425781 526.0316772460938 498.9816589355469 C
    511.18011474609375 491.5242919921875 490.0733642578125
    503.84332275390625 512.224365234375 536.9449462890625 C
    549.0176391601562 521.1896362304688 536.6615600585938
    514.050537109375 513.7171020507812 490.702880859375" stroke-
    width="2.0" fill="rgb(255, 255, 255)" opacity="0.66609687"/>
    <path d="M 391.5841369628906 179.8427734375 C
    405.9173583984375 158.2277069091797 388.2593688964844
    164.4803924560547 356.5292663574219 162.7732696533203 C
    323.517333984375 163.61631774902344 316.2157287597656
    165.74468994140625 334.00250244140625 183.86312866210938 C
    370.9042663574219 183.80703735351562 347.9760131835937 5
    161.361328125 321.5344543457031 181.3673553466797 C
    326.6772155761719 200.31649780273438 342.8227233886719
    186.2371826171875 391.5841369628906 179.8427734375" stroke-
    width="2.0" fill="rgb(1, 1, 1)" opacity="0.9884637"/>
  </g>
</svg>
```

PATH 221

Path

```
M 391.5841 179.8428 C 405.9174
158.2277 388.2594 164.4804 356.5293
162.7733 C 323.5173 163.6163
316.2157 165.7447 332 172 C 370.9043
183.807 347.976 161.3613 325 187 C
```

CONFIGURATION

X 317.249 · Y 149.137 · Width 136.1 · Height 80.98

[✓] Snap to Grid · Point Precision 3

[ ] Show Ticks · Interval 5

[✓] Fill · [ ] Preview · [ ] Minify output

PATH OPERATIONS

Scale X 1 · Scale Y 1 · Scale

Translate X 0 · Translate Y 0 · Translate

Origin X 0 · Origin Y 0 · Angle 0° · Rotate

Number of decimals 1 · Round

Convert to relative · Convert to absolute · Reverse · Optimize

COMMANDS

M	391.584	179.842				
C	405.917	158.227	388.259	164.480	356.529	162.773
C	323.517	163.616	316.215	165.744	332	172
C	370.904	183.807	347.976	161.361	325	187
C	326.677	200.316	342.822	186.237	391.584	179.842

Figure 7: Screenshots from an SVG editor showing path-level structure of generated tactile SVGs.

However, there are a few limitations to note. The generated output often suffers from **path proliferation**, where complex regions are decomposed into numerous small path segments; this increases visual clutter in the SVG editor and makes overall editing more cumbersome. Additionally, **residual background artifacts** sometimes remain after generation—these faint strokes must be manually identified and removed, adding extra post-processing overhead. Finally, the generated SVG frequently contains adjacent, disconnected paths that would benefit from being programmatically merged or simplified post-generation. By joining and streamlining these individual path segments, one could enhance both **tactile readability** and **editing convenience**.

In summary, while the generated SVGs provide excellent path-level control and modularity for fine edits, some manual cleanup and path consolidation are still required to fully prepare tactile graphics for production or tactile display.

6.2 RL-Guided Inference-Time Optimization for Tactile SVGs

This work demonstrates a practical and extensible pipeline for generating tactile-optimized SVG graphics from text prompts using a two-stage approach—Semantic Image Vectorization (SIVE) followed by Vectorized Particle Score Distillation (VPSD). Crucially, we avoid retraining the base diffusion model by using inference-time optimization with a lightweight LoRA head and a hybrid reward function composed of both visual (ImageReward) and tactile (rule-based) components. Our method illustrates that even minor stylistic changes in the SIVE stage—such as increased stroke width, contrast maximization, and background suppression—significantly improve tactile readiness before reinforcement-style refinement begins.

This end-to-end pipeline highlights the feasibility of steering diffusion-generated vector graphics toward accessibility goals using simple but effective techniques. The plug-and-play nature of our implementation also allows others to substitute custom tactile metrics, adjust reward weightings, and rapidly observe the resulting design trade-offs—all without modifying core diffusion weights.

6.3 Sensitivity to Hyperparameters and Reward Scaling

Our experiments emphasize that tactile-optimized SVG generation is highly sensitive to VPSD hyperparameters, particularly the number of inference-time iterations and the weighting of the tactile reward. Shorter VPSD runs (e.g., 500 steps) yield incomplete convergence with inconsistent stroke widths and overlapping paths. At 1,000 steps, however, the outputs balance visual structure with tactile simplicity—achieving clean silhouettes and spaced ridges suitable for haptic exploration. Pushing further to 5,000 steps causes overfitting, introducing excessive ridge detail and curve noise that degrade touchability.

Likewise, reward weighting impacts convergence behavior. Higher tactile emphasis (e.g., 70%) increases stroke consistency and spacing quality but may sacrifice some structural fidelity. Mid-range weights strike a useful balance, while excessive bias toward visual fidelity diminishes tactile clarity. These trade-offs underscore the value of inference-time control when optimizing for accessibility, enabling designers to tailor outputs based on use-case requirements.

6.4 Limitations and Future Directions

While our hybrid reward effectively enforces structural and perceptual constraints, it remains heuristic in nature. Rule-based metrics like stroke width variance and inter-path distance do not fully capture the nuances of human tactile perception. Fine lines or overly dense regions may still appear, and without real-world user studies, these metrics serve only as proxies. Furthermore, our method currently targets single-object prompts and simple illustrations. Extending this framework to multi-object STEM diagrams, charts, or circuit schematics would require advances in layout-aware vectorization and modular reward design.

In future work, we envision integrating semantic segmentation and object grouping into the vectorization process to enable tactile structuring of more complex scenes. Additionally, incorporating feedback from blind and low-vision users could help refine the tactile reward into a more perceptually grounded objective. With these improvements, the pipeline could serve as a foundational tool for producing educational materials that are both accessible and editable.

6.5 Broader Impact

By introducing RL-based tactile refinement into the vector graphic generation process, our approach represents a step toward democratizing the creation of inclusive visual content. The ability to generate haptically legible, structurally coherent SVGs from text enables new forms of rapid content prototyping in tactile media, especially in educational and assistive contexts. This method lays the groundwork for future systems that can support blind and low-vision learners with dynamic, interactive, and personalized diagrammatic content—advancing both AI-driven creativity and accessibility.

7 Conclusion

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

8 Team Contributions

Solo Project Seonghee Lee (shl1027).

Changes from Proposal Compared to the original proposal, the final project pivoted from RLOO-based fine-tuning of a pretrained SVG generation model (such as StarVector) toward a more lightweight, inference-time optimization strategy. Initially, we planned to directly fine-tune StarVector using reinforcement learning with a custom reward function (*TactileScore*), composed of semantic, structural, and compression-based components. However, due to time constraints, GPU resource limitations, and difficulties in stable RL training with token-level outputs, we instead focused on optimizing the SVG generation pipeline using the SVGDreamer framework. Specifically, we introduced a hybrid reward combining ImageReward (visual aesthetics) and rule-based tactile metrics (e.g., stroke consistency, spacing, and background simplicity), and applied this reward during the Vectorized Particle Score Distillation (VPSD) stage to guide inference-time refinement. This alternative allowed for more controlled experimentation with reward scaling and iteration counts while still exploring the core question of how reinforcement-style feedback can enhance SVGs for tactile accessibility.

References

Arash Ahmadian, Chris Cremer, Matthias Galle, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Ustun, and Sara Hooker. 2024. Back to Basics: Revisiting Reinforce-Style Optimization for Learning from Human Feedback in LLMs. *arXiv preprint arXiv:2402.14740* (2024).

American Printing House for the Blind. 2022. Guidelines and Standards for Tactile Graphics. `https://www.aph.org/product/guidelines-and-standards-for-tactile-graphics/`. Accessed: 2025-04-25.

Kevin Frans, Fanbo Xiang, and Igor Mordatch. 2023. VectorFusion: Text-to-SVG by Abstracting Pixel-based Diffusion. arXiv:2303.08057 [cs.CV] `https://arxiv.org/abs/2303.08057`

Yiwen Hu, Hang Zhou, Wayne Wu, Ziwei Liu, and Chen Change Loy. 2023. IconShop: Text-Guided Icon Editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. 2020. Differentiable Vector Graphics Rasterization for Editing and Learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 6 (2020), 193:1–193:15.

Guotao Liang et al. 2024. OmniSVG: Scaling up SVG Representation for Vector Graphic Generation. arXiv:2403.13312 [cs.CV] `https://arxiv.org/abs/2403.13312`

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *International Conference on Machine Learning*.

Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. 2024. StarVector: Generating Scalable Vector Graphics Code from Images and Text. arXiv:2312.11556 [cs.CV] `https://arxiv.org/abs/2312.11556`

Peter Selinger. 2003. Potrace: a tool for tracing a bitmap into a smooth, scalable image. `http://potrace.sourceforge.net`. Accessed: 2025-04-25.

Ximing Xing, Juncheng Hu, Guotao Liang, Jing Zhang, Dong Xu, and Qian Yu. 2025. Empowering LLMs to Understand and Generate Complex Vector Graphics. arXiv:2412.11102 [cs.CV] `https://arxiv.org/abs/2412.11102`

Ximing Xing, Haitao Zhou, Chuang Wang, Jing Zhang, Dong Xu, and Qian Yu. 2024. SVGDreamer: Text Guided SVG Generation with Diffusion Model. arXiv:2312.16476 [cs.CV] `https://arxiv.org/abs/2312.16476`

Yan Zhu. 2022. VTracer: A modern image vectorizer written in Rust. `https://github.com/visioncortex/vtracer`. Accessed: 2025-04-25.

A Additional Experiments

A.1 SIVE Process Image Comparison Results

A.1.1 Example of a Good Tactile Graphic

A well-designed tactile graphic uses minimal background detail and clear outlines.

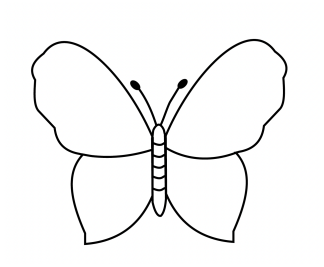


Figure 8: Example Tactile Graphic in the BANA guidelines: clear, thick lines and no extraneous detail.

A.1.2 Baseline Style

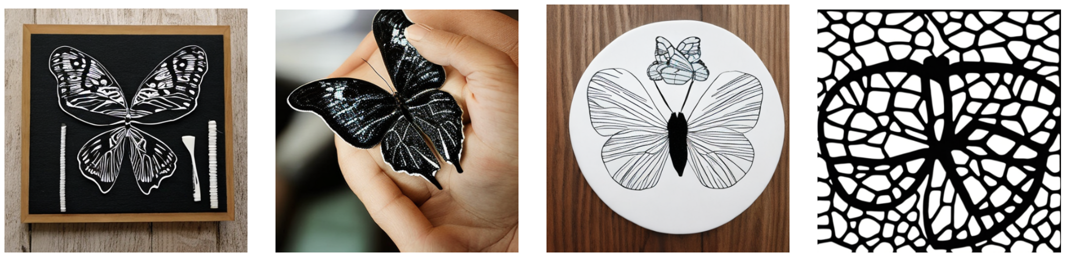
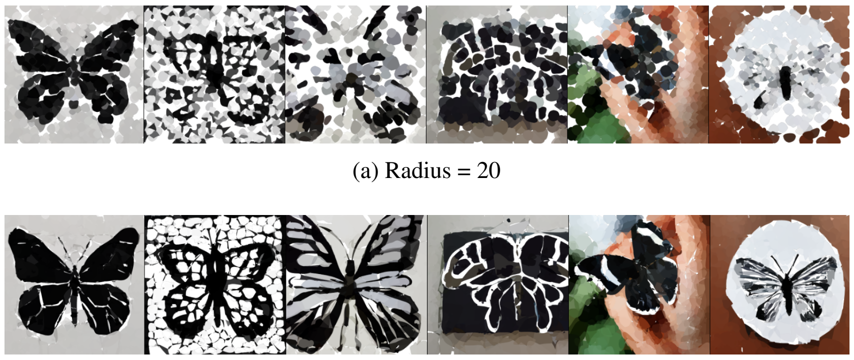

(a) Baseline 1 (b) Baseline 2 (c) Baseline 3 (d) Baseline 4

Figure 9: SVGDreamer outputs without the custom tactile style and just the given prompt: unwanted elements in the background and color artifacts appear.

A.1.3 Using Different Radius Sizes

Small particle radius yields finer, more precise vector “strokes,” suppressing background noise and malformed artifacts.


(a) Radius = 20

(b) Radius = 5

Figure 10: Comparison of SIVE initialization with (top) large vs. (bottom) small particle radius. Smaller radius produces cleaner object outlines and fewer spurious details.

Interpretation of Radius The particle radius in the SIVE stage controls the initial size of vector “strokes” used to approximate image structure. A larger radius (e.g., 20) captures broad shapes quickly but tends to overlook finer details, while a smaller radius (e.g., 5) enables more precise boundary alignment and eliminates spurious background textures.

Experiment	t	L_{total}	L_{lora}
VPSD without tactile	462.00	958.5000	0.0936
VPSD with tactile reward	351.00	504.0000	0.1831

Table 2: Comparison of key VPSD optimization metrics with and without the tactile-specific reward signal.

A.2 VPSD Process Tactile Reward Results

These results demonstrate that incorporating a tactile-specific reward into the VPSD optimization process significantly improves key performance metrics. The total loss Ltotal is reduced by nearly half, indicating more effective convergence and better overall optimization. Interestingly, the LoRA-specific loss increases, which suggests that the model is adapting more aggressively through the low-rank adapters to accommodate the tactile reward signal. This trade-off highlights that the tactile reward encourages meaningful structural adjustments rather than superficial refinements. Additionally, the optimization requires fewer steps (t=351, vs t=462), suggesting that the reward accelerates convergence by focusing the model on salient tactile features such as bold lines and simplified shapes. Overall, the tactile reward appears to steer the model towards producing outputs that are more interpretable by touch, albeit with some residual complexity that could be further addressed through post-processing or additional regularization.

The tactile reward guides the fine-tuning to prioritize line clarity and consistency, though some background clutter and over-detailing of the butterfly may persist.

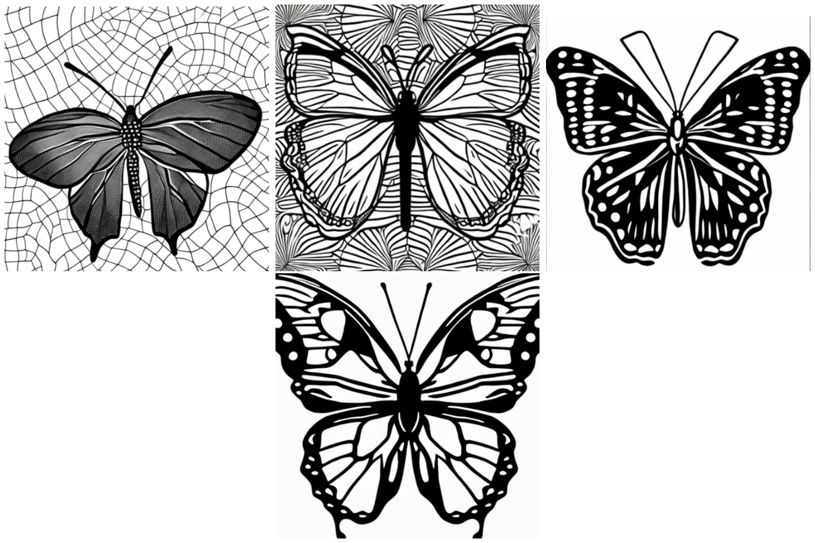
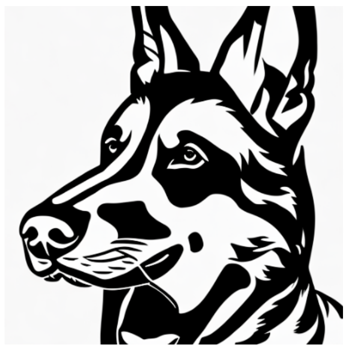


Figure 11: VPSD outputs with tactile reward enabled. Line strokes are more consistent, but further background simplification may be needed.

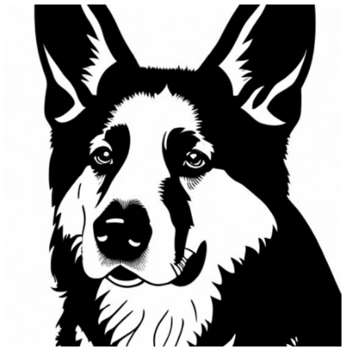
However, even after integrating the tactile reward into the VPSD stage, some outputs remain overly detailed—especially in complex prompts like animals or diagrams with textured regions. This suggests that further refinement is needed, potentially by integrating prompt-level complexity control or penalizing excessive detail directly in the reward function. For the final report I may explore prompt reweighting strategies or texture-suppressing penalties to enhance tactile suitability.

A.2.1 Other Prompts Tested: Husky Dog

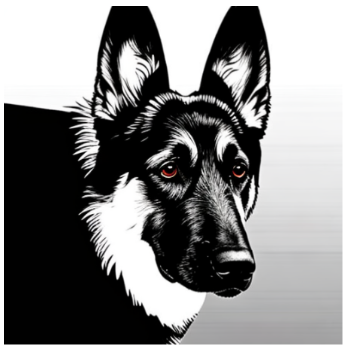
VPSD with the tactile reward also generalizes to subjects beyond simple diagrams, such as animals. We tested our pipeline using the prompt "a black-and-white husky dog contour line graphic" and observed that the tactile style effectively removes background clutter while preserving key facial structures. However, textured regions like fur often reintroduce high-frequency details, requiring additional prompt tuning or postprocessing. Figure 12 shows representative generations under this setting.



(a) Stylized contour with simplified muzzle and ears.



(b) Moderate detail; some interior fur textures persist.



(c) Overfit case with excessive stroke density.

Figure 12: Husky dog examples generated using tactile style and reward. Even without additional fine-tuning, the tactile pipeline generalizes reasonably well to animal prompts.

B Tactile Reward Implementation Details

The ‘TactileReward‘ class defines a hybrid reward function that combines visual aesthetic quality from ImageReward with domain-specific tactile metrics critical for haptic accessibility. This reward is used during the Vectorized Particle Score Distillation (VPSD) stage to guide the refinement of SVG outputs. The get_reward method receives a rendered image (either as a tensor, NumPy array, or PIL image), a list of DiffVG path objects representing the SVG geometry, and the original generation prompt. It computes the visual score via ImageReward and then calculates a tactile score based on three key metrics: stroke width consistency, path simplicity (via total control points), and a fixed spacing score. These components are linearly combined with a default weighting of 70% visual and 30% tactile to form the final reward. The weight of this Tactile reward was changed during the weight adjustment. This reward is differentiable through the rendering pipeline, enabling backpropagation into the latent space and producing outputs optimized for both appearance and tactile legibility.

Below is the Python implementation of the `TactileReward` class:

```
import torch
from PIL import Image
import numpy as np
import pydiffvg

# Import the original ImageReward
from ImageReward.ImageReward import ImageReward

class TactileReward:
    def __init__(self, reward_path='./checkpoint/ImageReward'):
        # Initialize the base ImageReward model
        self.base_reward = ImageReward(reward_path)
        self.device = self.base_reward.device

    def get_reward(self, rendered_img, svg_paths, prompt):
        """
        Calculate a hybrid reward combining ImageReward with tactile-
            ↪ specific metrics

        Args:
            rendered_img: The rendered PNG image of the SVG
            svg_paths: The vector paths from the SVG
            prompt: The text prompt used for generation
        """
        # Convert tensor to PIL image if needed
        if isinstance(rendered_img, torch.Tensor):
            img = rendered_img.squeeze(0).permute(1, 2, 0).cpu().numpy
                ↪ ()
            img = (img * 255).astype(np.uint8)
            pil_img = Image.fromarray(img)
        elif isinstance(rendered_img, np.ndarray):
            pil_img = Image.fromarray(rendered_img)
        else:
            pil_img = rendered_img

        # Get base aesthetic reward from ImageReward
        with torch.no_grad():
            base_reward = self.base_reward.score(prompt, pil_img)

        # Calculate tactile-specific metrics
        tactile_metrics = self._calculate_tactile_metrics(svg_paths)

        # Combine rewards with appropriate weights
        final_reward = base_reward * 0.7 + tactile_metrics * 0.3

        return final_reward

    def _calculate_tactile_metrics(self, svg_paths):
        """Calculate tactile-specific metrics from SVG paths"""
        # 1. Stroke width consistency
        stroke_widths = []
        for path in svg_paths:
            if hasattr(path, 'stroke_width'):
                stroke_widths.append(path.stroke_width.item())

        if stroke_widths:
            stroke_consistency = 1.0 - min(
                1.0,
                np.std(stroke_widths) /
                (np.mean(stroke_widths) if np.mean(stroke_widths) > 0
                    ↪ else 1)
            )
        else:
            stroke_consistency = 0.5

        # 2. Path simplicity
        total_points = sum(
            len(path.points) for path in svg_paths if hasattr(path, '
                ↪ points')
        )
        path_simplicity = 2.0 / (1.0 + np.exp(total_points / 200 -
            ↪ 1.5))

        # 3. Element spacing (simplified)
        spacing_score = 0.7

        # Combine metrics
        tactile_score = (
            stroke_consistency * 0.4 +
            path_simplicity     * 0.4 +
            spacing_score       * 0.2
        )

        return tactile_score
```

B.1 Prompts and Variables

The main prompt used for generating the tactile butterfly graphic was:

```
python svgdreamer.py \
  x=tactile \
  skip_sive=False \
  "prompt='A␣black-and-white␣butterfly␣tactile␣graphic␣contour␣outline
      ↪ ␣line␣graphic␣like␣a␣coloring␣book,␣created␣by␣tracing␣the␣
      ↪ outlines.␣The␣lines␣should␣be␣continuous,␣and␣uniform␣in␣
      ↪ width.␣There␣should␣be␣no␣background␣and␣no␣extraneous␣detail
      ↪ .␣Patterns␣on␣the␣wings␣should␣only␣emphasize␣key␣details.'"
      ↪ \
  result_path=./logs/tactile_butterfly \
  seed=262 \
  multirun=False \
  diffuser.download=True
```

The x argument selects the “tactile” style—a variant I derived by adapting the built-in styles for raised-dot graphics. I drastically reduce background iterations (since tactile graphics rely on clear, uncluttered outlines), force a strictly black-and-white palette with finer stroke radius, and allocate more iterations to foreground extraction so key features (wing veins, body segments) receive sufficient vector refinement. I also enforce ample spacing between strokes during initialization and reinitialization to ensure all elements meet minimum tactile separation guidelines, resulting in SVGs that are both machine-optimizable and highly legible by touch.

C GitHub Repository Guide

The full implementation and experimental logs are available at: `https://github.com/shljessie/tactile-svgdreamer`.

C.1 Repository Structure

- `main/`
  - `svgdreamer/`: Core SVGDreamer codebase.
  - `ImageReward/`: Pretrained ImageReward model wrapper.
  - `conf/`: Configuration YAML files for SIVE and VPSD stages.
  - `logs/`: Experiment logs and output SVGs organized by prompt and hyperparameters.
  - `scripts/`: Utility scripts (e.g., training launchers, result renderers).
  - `data/` (if included): Prompt templates and SVG examples.
- `checkpoint/`: Saved ImageReward weights and trained LoRA U-Net models.
- `README.md`: Setup instructions, dependencies, and quickstart guide.

C.2 Quickstart

1. **Clone the repo:**

```
git clone https://github.com/shljessie/tactile-svgdreamer.git
cd tactile-svgdreamer
```

2. **Create a conda environment:**

```
conda env create -f environment.yml
conda activate tactile-svgdreamer
```

3. **Download pretrained weights:** Place `ImageReward.pt` into `checkpoint/`. Optionally, populate `checkpoint/lora/` with pretrained LoRA U-Net checkpoints.
4. **Run SIVE stage:**

```
python scripts/run_sive.py \
  --config conf/sive_tactile.yaml \
  --prompt "a black-and-white butterfly tactile graphic ..." \
  --output logs/tactile_butterfly/
```

5. **Run VPSD + Hybrid Reward:**

```
python scripts/run_vpsd.py \
  --config conf/vpsd_tactile.yaml \
  --checkpoint logs/tactile_butterfly/step_100.ckpt \
  --prompt "a black-and-white butterfly tactile graphic ..." \
  --output logs/tactile_vpsd_beta_0.5/
```

6. **Visualize Outputs:** Rendered images and SVGs are saved in the specified `output/` folders. Use:

```
python scripts/visualize.py --input logs/tactile_vpsd_beta_0.5/
```

   to generate figures for reports or slides.

C.3 Customizing Experiments

- To adjust reward weights (β), modify `reward_weight` in the VPSD config (e.g., 0.3, 0.5, 0.7).
- To change iteration count, set `num_iter` in the VPSD config.
- To try different prompts, edit or pass new strings via the `-prompt` argument.

C.4 Reproducing Results

All hyperparameters and folder structure align with those reported in this paper. You can reproduce key figures for the butterfly prompt by running the above commands with equivalent configs. Logs with suffixes `_1000`, `_5000`, etc. correspond to different iteration counts used in the experiments.

Notes

- Ensure GPU availability and correct CUDA drivers before executing the scripts.
- For detailed error tracking or debugging, examine the `logs/` directory for per-step loss metrics and intermediate SVG visualizations.