Appendix

7. Implementation Details

This section provides details on the implementation of our approach. We describe the curve generation process for the inscribed square problem (Sec. 7.1), the square enhancement procedure used to refine predictions (Sec. 7.2), and the instance generation pipelines for the Steiner tree (Sec. 7.3) and maximum area polygon (Sec. 7.4) problems. We conclude with details on the model architecture (Sec. 7.5) and training procedure (Sec. 7.6).

7.1. Curve Generation

For harmonic-based curves, we construct a random radial profile

$$r(\theta) = 1 + \sum_{h=1}^{H} \rho_h \sin(h\theta + \phi_h),$$

with H sampled uniformly from $[H_{\min}, H_{\max}]$ and amplitudes ρ_h drawn from a decaying envelope to produce smooth perturbations. Square vertices are first placed in Cartesian coordinates, then converted to polar coordinates (θ_i, r_i), revealing the radial mismatch between the vertices and the base profile (Fig. 10a). A periodic cubic spline is fit to the radius corrections $r_i - r(\theta_i)$, ensuring that the resulting contour passes exactly through all square vertices (Fig. 10b). To guarantee validity, we enforce periodicity in the spline domain and regenerate until a non-self-intersecting Jordan curve is obtained. A random global translation is then applied, and both the curve and its inscribed squares are normalized to fit inside $[-1, 1]^2$ before rasterization. An example of the final output is shown in Fig. 10c.

7.2. Square Enhancement

To extract the initial square $\widehat{S}$ from the predicted mask, we fit a contour-aligned minimum-area rectangle and take its four vertices as $V(\widehat{S}) = \{p_i\}_{i=1}^4$. For a candidate rigid transform (R_θ, t) with θ in radians and $t \in \mathbb{R}^2$, we form the transformed square

$$S(\theta, t) \;=\; R_\theta \widehat{S} + t,$$

with vertices $V(S(\theta, t)) = \{q_i(\theta, t)\}_{i=1}^4$, where $q_i(\theta, t) = R_\theta p_i + t$. We then evaluate the alignment score with the curve C as defined in Eq. 1.

The final snapped square is obtained by selecting the rigid transform that maximizes this score:

$$(\theta^*, t^*) = \arg\max_{\theta, t} \mathcal{A}(S(\theta, t), C), \qquad S^* = R_{\theta^*} \widehat{S} + t^*.$$

We approximate this maximization with a discrete grid search. Specifically, we sample $\theta \in [\theta_{\min}, \theta_{\max}]$ with step $\Delta\theta$, and translations $t = (\Delta x, \Delta y)$ with $\Delta x, \Delta y \in \{-T, \dots, T\}$ in steps of one pixel. For each candidate (θ, t), the square mask is rigidly warped, its corners recomputed, and $\mathcal{A}(S(\theta, t), C)$ evaluated.

7.3. Steiner Tree Generation

Generation of a synthetic instance begins by sampling n terminal nodes within the unit square, with n drawn uniformly from $[10, 20]$. To prevent rasterization artifacts, we enforce a minimum separation between terminals with rejection sampling. For each sampled configuration, we compute the Steiner Minimal Tree using the GeoSteiner solver [24]. The resulting solutions are rasterized into grayscale images of fixed resolution (128×128), where terminals and Steiner points are depicted as small filled black circles with a radius of 2 pixels and edges as thin white lines that are 2 pixels wide, while the background is gray. The final dataset contains 1,000,000 instances.

7.4. Maximum Area Polygon Generation

For MAXAP instance generation, we first sample n points within the unit square, with n drawn uniformly from $[7, 12]$. Also here we enforce a minimum separation between points with rejection sampling. For each set of point, we exhaustively go over all valid polygon configurations and find the one with the largest area. We employ a backtracking depth-first search to systematically explore all valid simple polygons formed by a given point set. The method fixes an anchor point at the bottommost-leftmost position to eliminate rotational symmetry, then incrementally constructs polygons by selecting vertices in angular order around the centroid. At each step, the algorithm prunes invalid branches by rejecting vertices that would create edge intersections with the existing partial polygon. When a complete polygon is formed, the closing edge is validated for intersections, and the polygon area is computed using the shoelace formula [33]. The search maintains the globally optimal solution by comparing areas and updating the best configuration found. This approach guarantees finding the maximum area simple polygon while significantly reducing the exponential search space through geometric pruning, making the $O(n!)$ worst-case complexity manageable and runtime that is fast in practice for small point sets ($n \leq 15$). Finally, the polygon is rasterized into grayscale images of fixed resolution (128×128), where the polygon edges are drawn as white lines that are 1 pixel wide, the polygon interior is black and the background is gray. In total the dataset contains 1,000,000 instances.

7.5. Model Architecture

Our approach employs a conditional diffusion model based on a U-Net architecture for generating geometric solutions. The U-Net consists of 4 encoder and decoder levels with

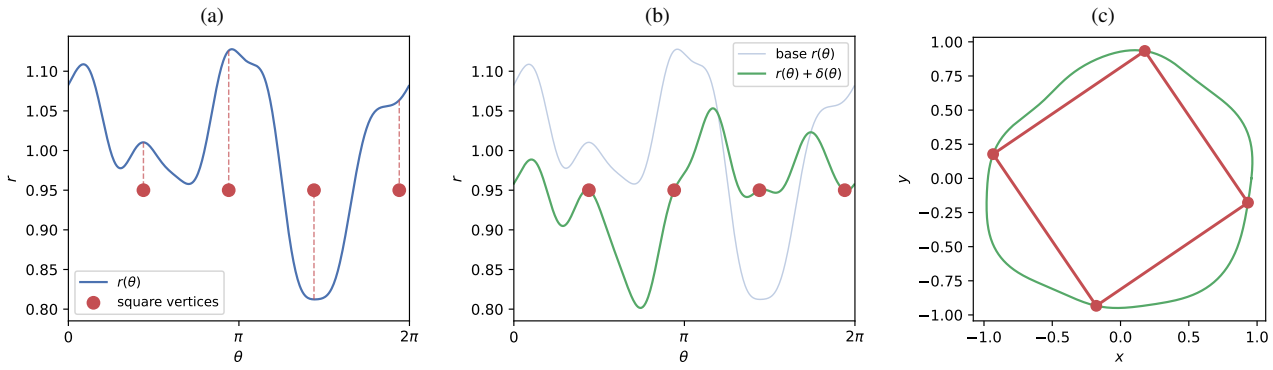

Figure 10. Curve generation pipeline for a single inscribed square for $H = 8$ terms. (a) Base harmonic radial profile $r(\theta)$ with square vertices (red) shown in polar coordinates; dashed lines indicate the radial mismatch. (b) After fitting a periodic cubic spline correction $\delta(\theta)$, the adjusted curve passes exactly through all four vertices. (c) Final closed curve in Cartesian coordinates with the inscribed square.

a base channel count of 64, following a standard channel progression of $64 \to 128 \to 256 \to 512$ in the encoder path. The model takes a 2-channel input (noisy target and condition images) and produces a single-channel denoised prediction.

Multi-head self-attention with 8 heads is integrated at the bottleneck and at encoder/decoder levels 2 and 3. The attention mechanism uses GroupNorm with 32 groups. Each level incorporates residual blocks with two 3×3 convolutional layers, BatchNorm, and ReLU activations, along with time embedding injection through learned linear projections. Sinusoidal time embeddings with 128 dimensions condition the model on the diffusion timestep.

7.6. Training Procedure

The model is trained using the DDIM (Denoising Diffusion Implicit Models) framework with 100 diffusion steps, a linear beta schedule and deterministic sampling ($\eta = 0.0$). We employ an L_2 loss on noise prediction, where the model learns to predict the noise ϵ added to the clean image at timestep t:

$$\mathcal{L} = \mathbb{E}_{t,\epsilon\sim\mathcal{N}(0,I)}\left[\|\epsilon - \epsilon_\theta(x_t, t, c)\|_2^2\right]. \tag{3}$$

Training is performed using the AdamW optimizer with a learning rate of 6×10^{-4} and cosine annealing with warm restarts over 0.5 cycles, including 100 warm-up steps and a minimum learning rate factor of 0.1. We use gradient accumulation over 8 steps with gradient clipping at a maximum norm of 1.0, and a batch size of 128 per GPU. The training employs mixed precision with bfloat16 autocast for efficiency and is distributed across 4 NVIDIA GTX 3090 GPUs for 100 epochs.

8. Additional Experiments

In this section we present additional experiments and ablations that complement the main evaluation. We first evaluate inscribed square predictions in parametric space to quantify the approximation error introduced by our pixel-space formulation (Section 8.1), then compare our diffusion framework against a regression baseline (Section 8.2), and finally compare against prior learning-based methods on MaxAP and Steiner Tree (Section 8.3).

8.1. Parametric Evaluation for Inscribed Squares

To evaluate the effect of operating in pixel space rather than in the original continuous domain, we additionally evaluate our method in parametric space, where squares are represented by their four corner coordinates and curves by their original polyline vertices. This allows us to measure the true geometric accuracy of our predictions, independent of rasterization.

Given a predicted square, we extract its four corners from the output image. Table 4 presents results in both evaluation settings. In pixel-space evaluation, all metrics are computed on rasterized images: squareness from the rendered square mask, and alignment via integer lookups in the distance transform (reproduced from Table 1 for reference). In parametric-space evaluation, we compute squareness directly from the float corner coordinates and alignment as the mean point-to-polyline distance to the continuous curve.

In parametric space, GT achieves perfect scores by construction. GT (raster) rasterizes the ground-truth curve and square, then extracts corners back to continuous coordinates, quantifying the round-trip cost of discretization and serving as an upper bound.

Importantly, our predictions maintain similar relative performance to the upper bound in both evaluation settings. Snapping improves alignment in both cases. These results confirm that the pixel-space formulation preserves geometric accuracy and that discretization error does not compound with the model's prediction error. We note that both sources of error can potentially be reduced, discretization through higher resolution and prediction error through

Table 4. Pixel-space vs. parametric-space evaluation for inscribed squares. GT refers to the ground-truth square corners; GT (raster) shows the effect of rasterizing the ground truth. Pixel-space results are reproduced from Table 1.

	Squareness ↑	Alignment ↑ (px)
Pixel-space evaluation		
GT	0.924	-0.14
Pred	0.892	-1.60
+ snap	0.891	-0.90
Parametric-space evaluation		
GT	1.000	0.00
GT (raster)	0.919	-0.27
Pred	0.880	-1.65
+ snap	0.879	-1.17

Table 5. **Regression Model Evaluation Results.** Comparison of diffusion (best-of-10) and regression models. Reported are valid polygon rates and mean area ratios (± std) across input point ranges.

Input Points	Diff. Valid Rate	Reg. Valid Rate	Diffusion Ratio Mean ± Std	Regression Ratio Mean ± Std
7-12	0.953	0.361	0.988 ± 0.020	0.9994 ± 0.0025
13-15	0.620	0.016	0.962 ± 0.041	0.9988 ± 0.0031

increased model capacity or training data. We leave such exploration to future work.

8.2. Regression Model Ablation

For certain geometric problems, each input instance admits a unique optimal solution. In such cases, one may wonder whether training a diffusion model is necessary, since the task appears to reduce to learning a deterministic mapping. Indeed, the conditional distribution to be learned degenerates into a collection of Dirac delta functions.

To examine this, we compared our diffusion framework with a direct regression baseline. For both the Maximum Area Polygon and Steiner Tree problems, we trained a regression model with the same U-Net backbone as our diffusion model, using identical training data and budget. The regression model outputs a solution image in a single forward pass, conditioned only on the rasterized input instance.

For Maximum Area Polygon, the regression model succeeds on simpler instances but often produces polygons with blurry edges or missing segments for more complex cases. This degrades our polygon extraction stage, which frequently fails to recover a valid polygon from such outputs. The quantitative results, shown in the "Reg. Valid Rate" column of Table 5, confirms this limitation.

We observe similar behavior for the Steiner Tree Problem (Table 6). The regression baseline produces valid trees at substantially lower rates, with validity dropping sharply as the number of points increases. When valid trees are produced, the length ratios are comparable to diffusion, but the regression model fails on the majority of complex instances.

Table 6. Regression vs. diffusion ablation for Steiner Tree Problem. We report valid tree rates and mean Euclidean length ratios (± std) relative to the optimal solution.

Input Points	Diff. Valid Rate	Reg. Valid Rate	Diffusion Ratio Mean±Std	Regression Ratio Mean±Std
10-20	0.996	0.727	1.0008±0.0005	1.002±0.006
21-30	0.986	0.344	1.0018±0.0011	1.003±0.006
31-40	0.834	0.072	1.0044±0.0035	1.006±0.008
41-50	0.334	0.007	1.0092±0.0055	1.013±0.008

Table 7. Comparison with DIFUSCO on Maximum Area Polygon.

Points	Method	Validity ↑	Area Ratio ↑	Optimal ↑
7–12	DIFUSCO	95.2%	0.997	75.2%
	Ours	95.3%	0.989	57.4%
13–15	DIFUSCO	66.5%	0.991	20.0%
	Ours	62.0%	0.962	6.2%

In contrast, the diffusion model retains a key advantage: stochasticity. By conditioning on both the input instance and a noise vector, we can generate multiple candidate solutions and resample until a valid output is obtained. This property directly improves the robustness of the approach and highlights why diffusion remains beneficial even in problems with a single optimal solution.

8.3. Comparison with Learning-Based Methods

We compare our approach against representative prior learning-based solvers that apply diffusion models to geometric optimization problems.

Maximum Area Polygon. We compare against DIFUSCO [42] on the MaxAP problem, evaluating both methods using best-of-10 sampling. For a fair comparison, DIFUSCO is evaluated using top-n edge selection without greedy insertion, consistent with our evaluation protocol. Results are shown in Table 7.

DIFUSCO performs slightly better on MaxAP with comparable validity rates. However, DIFUSCO relies on domain-specific architectures tailored to combinatorial optimization and does not extend to the Inscribed Square or Steiner Tree problems, while our approach handles all three identically using a standard visual diffusion model.

Table 8. Comparison with DeepSteiner on Steiner Tree Problem. Length ratio to optimal (lower is better).

Method	10–20 pts	21–30 pts	31–40 pts	41–50 pts
MST	1.0363	1.0416	1.0470	1.0522
DeepSteiner	1.0355	1.0413	1.0466	1.0517
Ours	1.0008	1.0018	1.0044	1.0092
Ours Valid Rate	99.6%	98.6%	83.4%	33.4%

Steiner Tree. We compare against DeepSteiner [48] on the Steiner Tree Problem. DeepSteiner is trained on instances with 10–20 points. We report the length ratio to optimal, where lower is better. Results are shown in Table 8.

While DeepSteiner achieves perfect validity by construction since it starts from an MST solution and refines it, its results remain only marginally better than the MST baseline. In contrast, our approach produces approximations substantially closer to optimal, even when extrapolating to instances with more points than seen during training. The tradeoff is reduced validity at higher point counts, reflecting the increased difficulty of generating valid tree structures in pixel space.