MamTiff-CAD: Multi-Scale Latent Diffusion with Mamba+ for Complex Parametric Sequence

Supplementary Material

1. ABC-256 Dataset

We use the model links from the ABC [15] dataset to extract commands from the CSG representations of CAD models via the Onshape API [31] and FeatureScript, and convert them into parameterized sequences. During the filtering process, we retain only models with complete design operations and discard those with only sketching and extrusion operations, ensuring that the final command sequences have lengths ranging from 60 to 256. We conducted a statistical analysis of the CAD command sequence lengths (see Figure 7), which details the distribution of command sequence lengths in the training data. Notably, our dataset, with sequence lengths of 60–256, is currently the longest available CAD command sequence dataset.

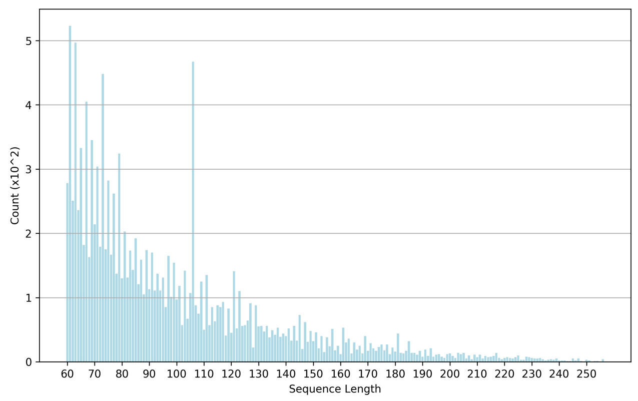
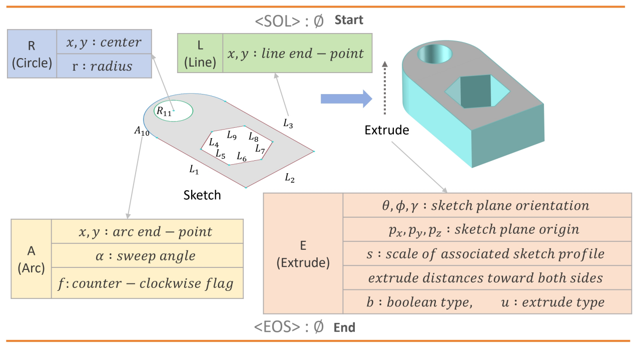


Figure 7. Distribution of CAD command sequence lengths in the ABC-256 dataset.

2. Parameterization Details

Figure 8 illustrates the complete command parameter list $p_i = [x, y, \alpha, f, r, \theta, \phi, \gamma, p_x, p_y, p_z, s, e_1, e_2, b, u] \in \mathbb{R}^{16}$, which describes the geometric information of each command in a CAD model. To ensure bounded values, each CAD model is first scaled into a $2 \times 2 \times 2$ cube (without translation), which confines the sketch plane origin (p_x, p_y, p_z) and extrusion distances (e_1, e_2) to $[-1, 1]$, the sketch profile scale s to $[0, 2]$, and the plane directions (θ, ϕ, γ) to $[-\pi, \pi]$. Moreover, sketch profiles are normalized to a unit square with the starting point fixed at $(0.5, 0.5)$, ensuring that the curve endpoint (x, y) and circle radius r lie in $[0, 1]$, and the arc sweep angle α in $[0, 2\pi]$. All continuous parameters are discretized into 256 levels using 8-bit integers, while discrete parameters remain unchanged. In a CAD model, sketch commands (initiated by $\langle\text{SOL}\rangle$) and extrusion commands alternate; the former define 2D closed profiles, and the latter extrude these profiles to form 3D solids with specified Boolean operations. Each command is represented as a 16×1 vector (unused entries set to -1), and the sequence length is fixed at $N_c = 256$ (padded with $\langle\text{EOS}\rangle$). This parameterization is similar to that in DeepCAD[42].



<SOL> : ∅ Start
R (Circle)
x, y : center
r : radius
L (Line)
x, y : line end − point
R11
A10
L3
L4 L9 L8 L7
L5 L6
L1
L2
Sketch
Extrude
A (Arc)
x, y : arc end − point
α : sweep angle
f : counter − clockwise flag
E (Extrude)
θ, ϕ, γ : sketch plane orientation
px, py, pz : sketch plane origin
s : scale of associated sketch profile
extrude distances toward both sides
b : boolean type, u : extrude type
<EOS> : ∅ End

Figure 8. CAD commands and their parameters. $\langle\text{SOL}\rangle$ indicates the start of a loop, $\langle\text{EOS}\rangle$ indicates the end of the entire sequence.

3. Model Architecture and Training Details

Autoencoder. The encoder is composed of four Mamba+ blocks. The state-space model has a feature dimension of 256, a convolution kernel size of 4, and an attention head dimension of 32. The decoder consists of four Transformer blocks, each containing 8 attention heads and a feed-forward network with a dimension of 1024. All modules employ standard layer normalization and a dropout rate of 0.1. The encoder compresses the input sequence into a latent space of dimension 64. At the final stage, the decoder uses two independent linear mappings to predict the command type and the command parameters. The command type prediction produces an output with dimensions 256 by 6 while the command parameter prediction yields an output with dimensions 256 by 4096, which is reshaped into a matrix of size 16 by 256. Training uses the AdamW optimizer with a weight decay set to 1e-4, an initial learning rate of 1e-3, and 2000 warmup steps. The batch size is 32 and the model is trained for 300 epochs with gradient clipping at a threshold of 1.0.

MST-Diffusion Generator. The diffusion generator adopts a linear diffusion strategy with 1000 steps. The model is trained for 200000 iterations with a batch size of 64. At each step the model predicts noise using mean squared error loss and is optimized by the Adam optimizer with a beta one value of 0.9. The initial learning rate is set to 2e-4 and the learning rate decays after 100000 iterations with a decay factor of 0.1. The diffusion model configuration is consistent with the autoencoder, with a latent vector dimension of 64, a sequence length of 256, an embedding dimension of 512, 6 Transformer layers, 8 attention heads, and a dropout rate of 0.1.

4. The Algorithm of Mamba+

As shown in Figure 3, the autoencoder architecture incorporates Mamba+ blocks, which utilizes a dual-branch structure to balance local feature extraction and long-range dependency modeling. The input embedding sequence $E_x \in \mathbb{R}^{B\times W\times D}$ is first decomposed by a linear projection into two branches: a feature transformation branch (b1) and a gating branch (b2). This process produces the main-path feature x and the gating signal z, respectively. In branch b1, a one-dimensional convolution followed by a SiLU activation enhances local features, yielding x'. Based on x', the state-space module parameters B and C, as well as the discretization factor Δ, are computed. In conjunction with a learnable parameter A, the discretization process generates $\bar{A}$ and $\bar{B}$, which are then fed into the state-space model (SSM) to compute h_{SSM}. Meanwhile, branch b2 applies the Sigmoid function to generate the gating signal G_{b2}; its complement, $G_f = 1 - G_{b2}$, acts as a forget gate over the historical feature x' to produce the adjusted feature x''. Finally, the output is obtained by fusing x'' with h_{SSM} and applying a linear projection to produce E_y, thereby completing the multi-level encoding of the input command sequence. The pseudocode for the Mamba+ algorithm is presented in the following table:

Algorithm: Mamba+ Block Implementation

```
Input: Embedded command sequence $E_x : (B, W, D)$
Output: Encoded features $E_y : (B, W, D)$
1:  $x \leftarrow \text{Linear}^x(E_x)$    // Feature branch ($b1$)
2:  $z \leftarrow \text{Linear}^z(E_x)$    // Gating branch ($b2$)
3:  $x' \leftarrow \text{SiLU}(\text{Conv1D}(x))$    // Local enhancement
4:  $A \leftarrow \text{Parameter}^A$    $(E\times N)$
5:  $B \leftarrow \text{Linear}^B(x'),\ C \leftarrow \text{Linear}^C(x')$
6:  $\Delta \leftarrow \log(1+\exp(\text{Linear}^\Delta(x'))) + \text{Parameter}^\Delta$
7:  $\bar{A}, \bar{B} \leftarrow \text{discretize}(\Delta, A, B)$    // Discretization
8:  $h_{\text{SSM}} \leftarrow \text{SSM}(\bar{A}, \bar{B}, C)(x')$    // State-space computation
9:  $\text{G}_{b2} \leftarrow \sigma(z),\ \text{G}_f \leftarrow 1 - \text{G}_{b2}$    // Forget gate
10:   $x'' \leftarrow \text{G}_f \cdot x'$    // Historical feature retention
11:   $h_{\text{out}} \leftarrow x'' + h_{\text{SSM}}$    // Feature integration
12:   $E_y \leftarrow \text{Linear}^{y'}(h_{\text{out}})$
13:   return $E_y$
```

5. Detailed Diffusion Model Architecture

In this work, we employ a diffusion model guided by a multi-scale Transformer to predict the noise at each diffusion timestep. The model accepts a noisy latent variable $Z_t \in \mathbb{R}^{N\times T\times D}$ and outputs an estimate of the injected noise ϵ. A time-conditioning mechanism maps each discrete diffusion step $t \in \{1, \ldots, T\}$ to a continuous embedding $\tau(t)$, which is then processed by an MLP to produce two sets of scale and shift parameters $\{\xi_1^l, \psi_1^l, \omega_1^l,\ \xi_2^l, \psi_2^l, \omega_2^l\}_{l=1}^L$, where L is the number of Transformer layers. In particular, ξ_i^l and ψ_i^l modulate the layer-normalized activations, whereas ω_i^l adjusts the strength of the residual connections.

Multi-Scale Transformer. Within each Transformer layer, we introduce three parallel attention branches to capture local, mid-range, and global dependencies. For a position i in the sequence, we define:

$$\begin{aligned} W_l(i) &= \{\, j \mid |i-j| \le 64\},\\ W_m(i) &= \{\, j \mid |i-j| \le 128\},\\ W_g(i) &= \{\, 1, 2, \ldots, T\}. \end{aligned} \tag{11}$$

We then construct a mask $M_w \in \mathbb{R}^{T\times T}$ (for $w \in \{l, m, g\}$) such that $M_w(i,j) = 0$ if $j \in W_w(i)$ and $-\infty$ otherwise. Each attention branch operates on its respective window, yielding outputs H_l, H_m, H_g. We then fuse these via a learnable gating mechanism:

$$H_{\text{fuse}} = W_o\Big[\sigma\big(W_g[\, H_l \parallel H_m \parallel H_g]\big) \odot \big(H_l \parallel H_m \parallel H_g\big)\Big], \tag{12}$$

where $\parallel$ denotes concatenation, σ is the sigmoid function, $\odot$ is elementwise multiplication, and W_g, W_o are learnable parameters.

Sequence-Aware Positional Encoding. To preserve the sequential logic of CAD commands, we augment each token embedding $Z \in \mathbb{R}^{T\times d}$ with a learnable scalar η. We define:

$$\text{PE}(Z)_{pos,j} = Z_{pos,j} + \eta \times \begin{cases} \sin\Big(\frac{\text{pos}}{10000^{\,j/d}}\Big), & \text{if } j \text{ is even,}\\ \cos\Big(\frac{\text{pos}}{10000^{\,(j-1)/d}}\Big), & \text{if } j \text{ is odd.} \end{cases} \tag{13}$$

Let ϕ_l denote the l-th Transformer layer (incorporating multi-scale attention and a feed-forward network). The model estimates the noise via:

$$\epsilon_\theta = W_{\text{out}} \circ \phi_L \circ \cdots \circ \phi_1\Big(\text{PE}\big(W_{\text{in}}\, Z_t\big) + \tau(t)\Big). \tag{14}$$

Inside each layer, we apply layer normalization and inject the time-derived parameters $\{\xi_i^l, \psi_i^l, \omega_i^l\}$ to modulate the computations. Denoting the layer input by X:

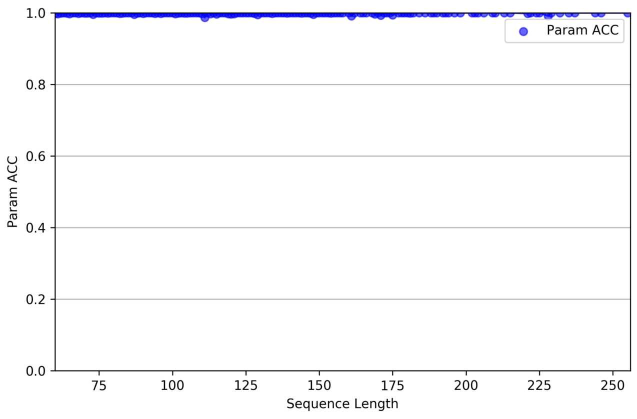


Figure 9. CAD command sequence length and its corresponding parameter accuracy.

$$\begin{aligned} X' &= \mathrm{LayerNorm}(X) \odot (1 + \xi_1^l) + \psi_1^l, \\ X_{\mathrm{attn}} &= X + \omega_1^l \odot \mathrm{Attention}(X'), \\ X'' &= \mathrm{LayerNorm}(X_{\mathrm{attn}}) \odot (1 + \xi_2^l) + \psi_2^l, \\ X_{\mathrm{out}} &= X_{\mathrm{attn}} + \omega_2^l \odot \mathrm{FFN}(X''). \end{aligned} \tag{15}$$

Forward Diffusion. We apply a linear variance schedule $\{\beta_t\}$:

$$\begin{aligned} \beta_t &= 0.0001 + 0.0199 \frac{t}{T}, \\ Z_t &= \sqrt{\alpha_t}\, Z_0 + \sqrt{1-\alpha_t}\, \epsilon, \\ \alpha_t &= \prod_{s=1}^{t} (1 - \beta_s), \end{aligned} \tag{16}$$

where $\epsilon \sim \mathcal{N}(0, I)$. The network is trained to predict ϵ by minimizing:

$$\mathcal{L} = \mathbb{E}_{t,\, Z_0,\, \epsilon}\Big[\|\epsilon - \epsilon_\theta(Z_t,\, t)\|^2\Big]. \tag{17}$$

Sampling Process. Starting from $Z_T \sim \mathcal{N}(0, I)$, we iteratively denoise:

$$Z_{t-1} = \frac{1}{\alpha_t}\Big(Z_t - \beta_t \frac{1}{\sqrt{1-\alpha_t}}\, \epsilon_\theta(Z_t,\, t)\Big) + \beta_t\, z, \tag{18}$$

where $z \sim \mathcal{N}(0, I)$. Here, α_t and β_t denote the attenuation factor and variance at step t. This iterative process gradually removes noise to recover a clean latent Z_0, enabling faithful reconstruction of the CAD command sequences.

6. Failure Cases

Although our model performs very well on the reconstruction task as shown in Figure 9, maintaining high accuracy on both short and long sequences and achieving parameter prediction accuracy consistently near 0.99, non-watertight objects are occasionally generated during the generation process. Figure 10 presents several failure examples, and these failures are mainly caused by limitations in the generator's process.

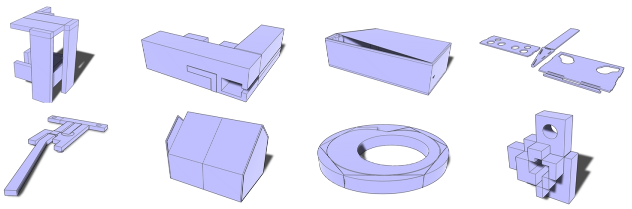


Figure 10. A collection of generated failed CAD models.

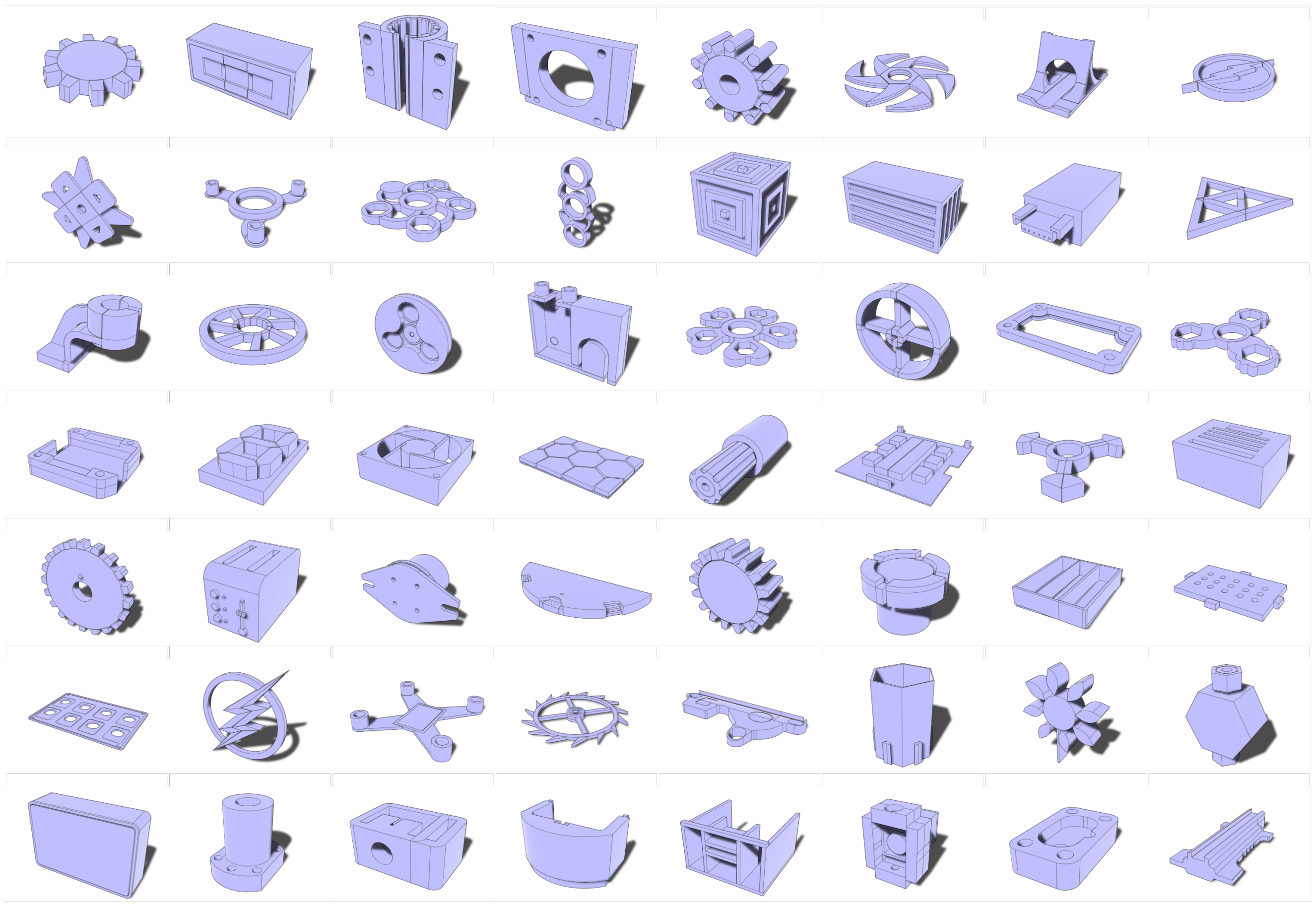


Figure 11. Unconditionally Generated CAD Models