VQ-SGen: A Vector Quantized Stroke Representation for Creative Sketch Generation

Supplementary Material

In this supplemental material, we provide further details on dataset processing, technical design, codebook exploration, and additional discussions.

A. Data Preprocessing and Augmentation

To enhance the model's performance and improve the quality of the training data, we adopt a series of preprocessing steps as shown in Fig. A1. Firstly, strokes that have the 'details' part label (*e.g.*, the sun, the ground, and the decorative dots) are moved, as these may introduce noise or unnecessary complexity. Secondly, excessively long strokes are eliminated to maintain a balanced distribution of stroke lengths (*e.g.*, the over-sketched pattern in the tail of the third creature), preventing outliers from disproportionately influencing the model. Lastly, shorter strokes that are connected are merged into a single stroke to maintain a better stroke structure. For instance, the wing in the second image (highlighted with a red box) is composed of multiple connected strokes that are merged into a single stroke, which ensures more coherent and natural groupings within the sketches.

Besides preprocessing, we employ a comprehensive augmentation strategy at both the stroke and sketch levels. At the stroke level, transformations such as rotation, translation, or scaling are applied to one or more randomly selected strokes. These operations diversify the stroke representations and improve the model's ability to generalize to various configurations. At the sketch level, the entire sketch is subjected to global transformations, including rotations or random removal of specific strokes, simulating incomplete or imperfect inputs.

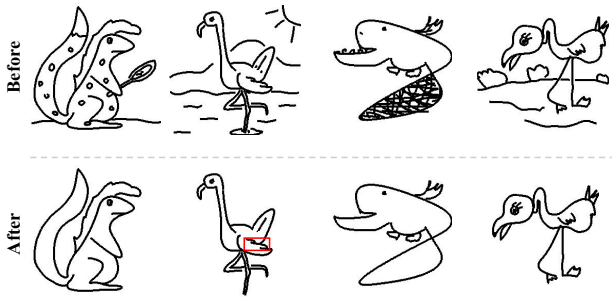


Figure A1. The demonstration of our data preprocessing.

B. Training and Inference Details

Implementation details. We use Adam optimizer [1] for training both the VQ-Representation and Gen-Transformer, while the learning rates are set as 10^{-4} for the former and 10^{-5} for the latter. We use step decay for VQ-Representation with a step size equal to 10 and do not apply learning rate scheduling for Gen-Transformer. We train our network on 2 NVIDIA V100 GPUs. The VQ-Representation network training takes around 20 hours with a batch size of 64. The Gen-Transformer is trained until convergence taking around 10 hours with a batch size of 8. The maximum stroke length N is 20 for Creative Birds and 35 for Creative Creatures to make sure it fits all the sketches. Stroke latent embedding $\boldsymbol{e}_i^s$ is a vector in $\mathbb{R}^{256}$. The number of codes V = 8192, and the code dimensions of both codebooks are 512. Stroke category number $C = 8$ for Creative Birds and 17 for Creative Creatures. The balancing weight $\alpha = 0.8$.

Training Strategy. We have three training stages. In the first stage, we train the stroke latent embedding auto-encoder. Once it is trained, both the encoder and decoder are fixed. In the second stage, we train the vector quantization auto-encoders and the corresponding codebooks. Similarly, once the network is trained, the three components are fixed. In the last stage, we train the cascaded generation decoders together, with the help of the aforementioned networks and codebooks. Note that, we follow [2] to address the teacher-forcing gap.

Teacher Forcing Gap. Teacher-forcing strategy is widely used for Transformer training. However, it introduces the exposure bias issue by always feeding the ground truth information to the network at training time but exploiting the inferior prediction at testing time. To overcome this issue, we follow [2] to mix the predicted stroke information with the ground truth information. The ratio of the ground truth strokes gradually decreases from 1.0 as the training progresses.

Sampling at Inference. Starting with a special *STR* token, the inference process of our *VQ-SGen* generates a sketch by sequentially sampling each subsequent element of the stroke sequence until reaching an *END* token. Each prediction step consists of two stages: (1) generating a minimal set of codes for sampling, comprising the top options whose cumulative probabilities exceed a threshold p_n, while ignoring other low-probability codes; (2) performing random sampling within this minimal set based on the relative probabilities of the codes. This approach ensures diversity in the predicted code sequence while reducing the risk of sampling unsuitable or irrelevant codes.

C. Location Codebook Probing

Table A1. Statistical comparison and ablation studies of the hyperparameter setting of the location codebook.

Methods	Creative Birds				Creative Creatures				
	IoU(↑)	FID(↓)	GD(↑)	CS(↑)	IoU(↑)	FID(↓)	GD(↑)	CS(↑)	SDS(↑)
2048×512	0.912	17.43	17.54	0.47	0.901	20.34	15.75	0.51	1.72
4096×512	0.943	16.15	18.68	0.51	0.923	18.32	16.54	0.55	1.79
4096×1024	0.954	16.31	18.53	0.52	0.934	18.07	16.86	0.56	1.81
Ours(8192 × 512)	**0.963**	**15.78**	**18.92**	**0.53**	**0.957**	**17.61**	**17.42**	**0.57**	**1.86**

Similar to the shape codebook, we investigated the configuration of the location codebook, in terms of the effects of its hyper-parameter setting on its representation ability and sketch generation. For the codebook size, we choose 2,048, 4,096, or 8,192, while for the feature size, we choose 512 or 1,024. Quantitative and qualitative experiments as shown in Fig. A2 and Tab. A1, respectively.

For the codebook representation ability, we investigate it via its reconstruction. For any location triplet, we recover its bounding box representation. Fig. A2 displays the input and reconstructed bounding boxes. We overlay the input sketch for better visualization. As can be seen, varying the codebook size or the feature size indeed impacts the representation ability, while our final choice obtains the best reconstruction. Additionally, we employ the mean IoU (Intersection over Union) of bounding boxes as a statistical measurement in Tab. A1, where the best IoU is achieved from our final choice. As for the sketch generation, we only measure the statistical metrics as in Tab. A1. Not surprisingly, our choice is the best. Although increasing the codebook size further may potentially enhance generation quality, similar to the findings with the shape codebook, we have not yet confirmed whether a larger codebook consistently yields performance improvements. Therefore, selecting the appropriate codebook size requires balancing generation quality and computational cost and may depend on the specific requirements of the task. A similar conclusion is drawn as in the shape codebook analysis, *i.e.*, we did not enumerate all the configurations, and thus cannot guarantee the optimal configuration.

D. Creativity Comparison with Diffsketcher

As presented in Sec. 5 in the main paper, our method can be easily adapted to do text-to-sketch generation, which aligns not exactly the same but closer with the goal of Diffsketcher [6]. However, we argue that the sketches they generated possess high realism and conform to conventional imagination since they are trained on realistic images. Fig. A3 displays a comparison. For Diffsketcher, we use a common prompt template - 'A bird is [input text]', except [Just dance], for which we use 'A bird is *performing* [just dance]'. Different random seeds and prompt templates are tested, but their sketches do not differ too much.

As can be seen, our sketches, *e.g.*, the smiling bird or the falling bird, are much more plausible, conforming with input text and representing more creative concepts, while their corresponding sketches cannot faithfully explain the text, limited to conventional bird sketches. Besides, their sketches usually have background strokes due to a different goal in the generation, *e.g.*, the tree branches. Especially, the dancing person is wrongly generated by their model, since it was fooled by the word dance, and cannot link the concept of dance with a bird. On the other hand, our method is over *two magnitudes* faster (*i.e.*, $0.86s$ vs. $153s$), making us both efficient and effective in the text-driven sketch gen-

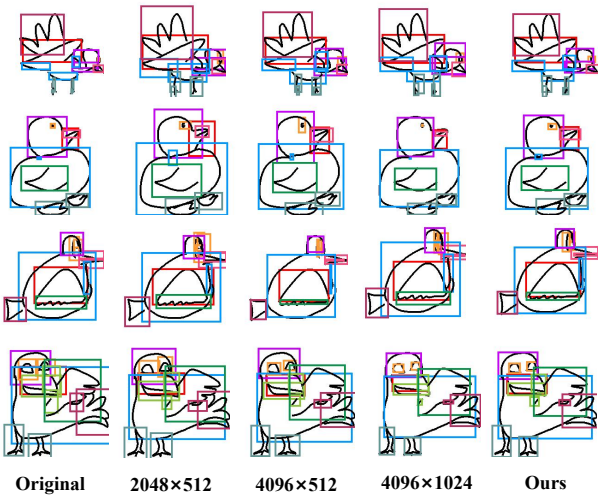


Figure A2. Visualization results of bounding box reconstruction. Note that the sketches are not generated, we overlay the input sketch to better perceive the bounding box placement.

Figure A3. The visual comparison with Diffsketcher [6] on the text-to-sketch application.

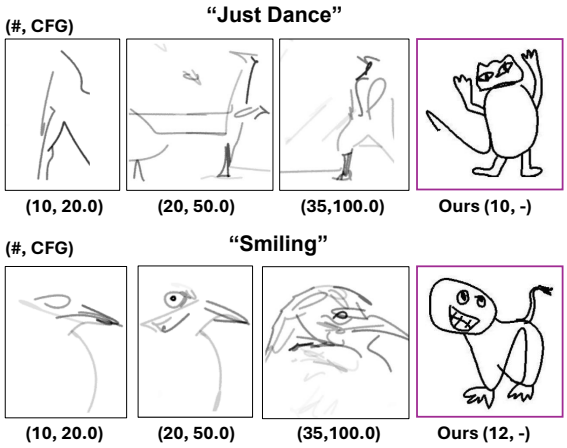


Figure A4. More DiffSketcher comparison [6] with varying numbers of strokes and CFG weights.

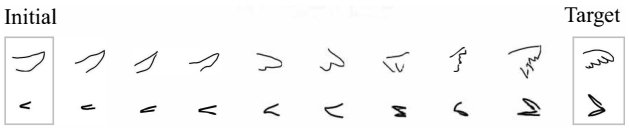


Figure A5. Two code interpolation examples, where the interpolation is executed from the initial to the target strokes (left to right).

eration task.

In order to generate more abstract sketches comparable to ours, two extra hyperparameters can be tuned in DiffSketcher - the number of strokes and the classifier-free guidance (CFG) weight. As shown in Fig. A4, we gradually increase the number of strokes as well as the CFG weight in two examples. When the stroke count is low (10 or 20 strokes), their sketches are too abstract to be recognizable. The smiling bird with 35 strokes starts focusing on realistic head details, but it falls short of creativity.

E. Further discussions

Code space interpolation. To further validate the clustering effects of the discrete code space, we conduct code interpolation experiments. Fig. A5 demonstrates two such examples. The interpolation is executed from the initial stroke to the target stroke (left to right). We first use linear α-blending to mix the two codes, and then fetch the closest code entry in $\mathcal{D}^s$ and decode it into a stroke image. We march 10 steps to achieve the target. As can be seen, the initial and target strokes can be meaningfully interpolated in the discrete code space.

Limitations. Our method has the following limitations: First, we did not extensively explore all combinations of codebook hyperparameters, so our current configuration may not be globally optimal. This could impact the quality of our model's generated results. For example, in Fig. A6(a), the wings exhibit stroke disconnection and blurring issues. Efficient hyperparameter search could potentially lead to improvements. Second, due to the decoupling of shape and position, we occasionally observe that the generated bounding box may fail to cover all strokes, resulting in a "hard cropping" of the outermost stroke. The strokes in the wings and head in Fig. A6(b) have experienced varying degrees of "hard cropping". Introducing a coverage regularizer could help address this issue. We leave the two as future work.

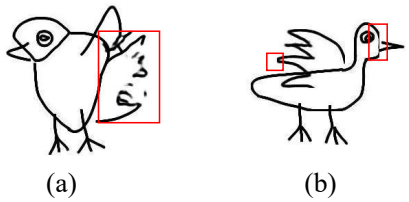


Figure A6. The demonstration of failure cases.

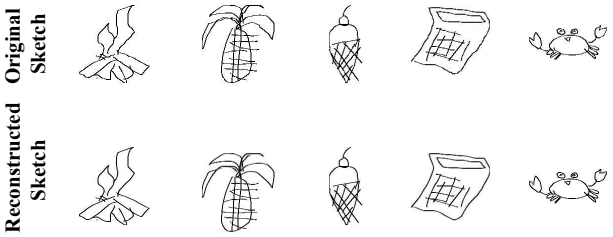


Figure A7. Sketch reconstruction on QuickDraw sketches.

Shape codebook generalization. Our learned shape codebook demonstrates strong generalization ability to unseen sketches from other datasets, even without the need for fine-tuning. This indicates that the learned representations capture fundamental structural patterns of sketches, enabling effective reconstruction across different data distributions.

Fig. A7 presents several examples of sketch reconstruction from the QuickDraw dataset [3]. As shown in the figure, our shape codebook successfully reconstructs diverse sketches with high fidelity, preserving key structural details and stroke arrangements. The ability to generalize across datasets highlights the robustness of our learned representations, suggesting their potential applicability in various sketch-based tasks, such as sketch recognition, generation, and editing. Furthermore, the high-quality reconstruction results indicate that our codebook effectively captures essential shape priors, making it a versatile tool for sketch-related applications without requiring additional domain-specific adaptations.

Point sequence encoding Following ContextSeg [5], we use stroke bitmap sequences rather than point sequences to represent sketches. The latter yields inferior reconstructions (see Fig.5 in [5]). We conducted further statistical analysis in our generation scenario (it is segmentation in Con-

textSeg). Specifically, we replace the CNN with Sketchformer [4] (a Transformer), and train the VQ-VAE and our generator. In terms of reconstruction, Sketchformer only yields Acc=0.44/Rec=0.32, which is far below ours (Acc=0.96/Rec=0.97). As for the generation, our advantages are more obvious (theirs vs. **ours**): 75.42 vs. **15.78** (FID↓), 12.31 vs. **18.92** (GD↑), 0.16 vs. **0.53** (CS, close to and >0.45 is better), confirming the advantage of bitmap encoding.

F. Detailed Network Configuration

Intuitively, the stroke shape embedding and the learning of the codebook could be achieved simultaneously. However, the network responsible for learning stroke shape embeddings needs to be applied individually to each stroke to effectively capture its structural details, whereas the learning of the codebook must consider the entire sequence of strokes, performing the compression in the sketch level. Consequently, achieving optimal performance is challenging when attempting to combine these tasks. Therefore, we propose to handle them separately. To this end, we first convert each stroke into a latent embedding and further compress them in the latent space.

Figure A8 illustrates the detailed structure of our network. The embedding network has an encoder-decoder structure, accepting the grayscale stroke image input augmented with x and y coordinate channels. Specifically, the encoder comprises a total of 10 layers, which are grouped into four blocks. Each block is characterized by distinct feature dimensions (*i.e.*, 64, 128, 256, and 512, respectively), resulting in a stroke embedding in $\mathbb{R}^{256}$. Both decoder branches share the encoder structure, working symmetrically to transform the stroke embedding into stroke reconstruction and the distance map as in [5].

We also illustrate the detailed network architecture of our tokenizing network, which is used for learning the shape or location codebook. The input is a feature matrix in $\mathbb{R}^{N\times 256}$ composed of shape or location embeddings within a sketch. A Conv1d layer is first employed for feature extraction, followed by a MaxPool layer or ConvTranspose1d layer to compress or expand the feature dimensions. Specifically, the encoder comprises 8 layers grouped into three blocks, each characterized by distinct feature dimensions (*i.e.*, 256, 256, and 512), resulting in a feature of $\mathbb{R}^{N\times 512}$. The "x3" and "x2" of each block in Fig. A8 mean that the block is repeated 3 or 2 times. Subsequently, the feature corresponding to each stroke is replaced with the nearest code and then fed into the decoder. Both decoder branches share the encoder structure. The final output aims to reconstruct the latent input.

References

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

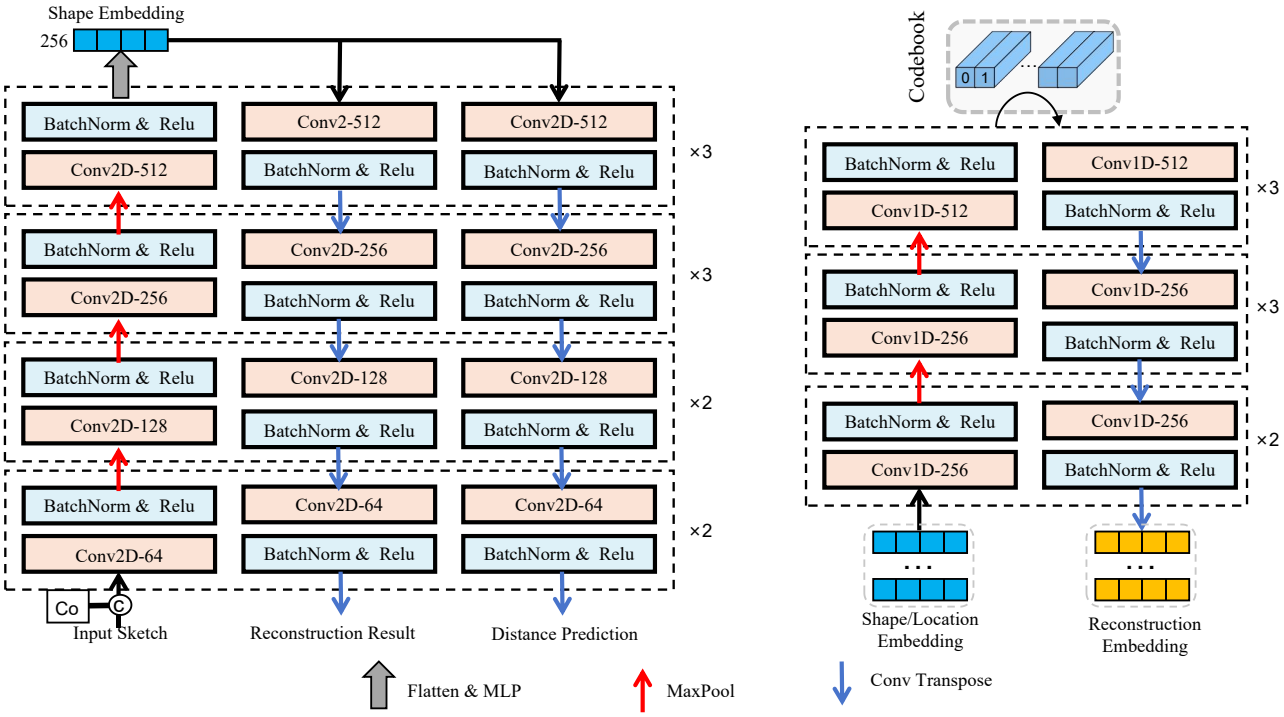
[2] Tsvetomila Mihaylova and André FT Martins. Scheduled sampling for transformers. *arXiv preprint arXiv:1906.07651*, 2019. 1

[3] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021. 3

[4] Leo Sampaio Ferraz Ribeiro, Tu Bui, John Collomosse, and Moacir Ponti. Sketchformer: Transformer-based representation for sketched structure. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14153–14162, 2020. 4

[5] Jiawei Wang and Changjian Li. Contextseg: Sketch semantic segmentation by querying the context with attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3679–3688, 2024. 3, 4

[6] Ximing Xing, Chuang Wang, Haitao Zhou, Jing Zhang, Qian Yu, and Dong Xu. Diffsketcher: Text guided vector sketch synthesis through latent diffusion models. *Advances in Neural Information Processing Systems*, 36:15869–15889, 2023. 2, 3



(a) Embedding Network

(b) Tokenizing Network

Figure A8. Detailed structure of our embedding networks and tokenizing network.