Supplementary Material

For paper "Multi-Vehicle Trajectories Generation for Vehicle-to-Vehicle Encounters"

I. NETWORK ARCHITECTURE

A. Multi-vehicle Trajectory Generator

Our neural network mainly contains two parts, including encoder and decoder. The basic architecture is built based on GRU. We will display the specific architecture in the following tables. Tab I displays the architecture and output dimension of encoder module. The whole sequence is processed at the same time and the final output is a embedding vector with $2 \times z_{dim}$. In our case, we set the dimension of $z_dim = 10$. After obtaining the embedding vector, we divide it into two parts. The first of it represents the mean value of z and the second part represents the standard deviation.

TABLE I

ENCODER OF MTG

Description		Output dimension
$Sequence_1(x_1, y_1)$	$Sequence_2(x_1, y_1)$	$50 \times 2, 50 \times 2$
Concatenated sequence (x_1, y_1, x_2, y_2)		50×4
Input layer (MLP)		50×64
ReLU		-
Encoder (GRU)		1×512[1]
Encoder (MLP)		512×20[2]
z_{mean}	z_{std}	$1 \times 10, 1 \times 10$

[1] We use bidirectional GRU here and the size of hidden state is 256, thus the size of output state is 256×2.

[2] The mean value and standard deviation are outputted in one embedding, and then will be divided into two parts.

TABLE II

DECODER OF MTG

Description		Output dimension
z	z	$1 \times 10, 1 \times 10$
Hidden layer (MLP)	Hidden layer (MLP)	$1 \times 256, 1 \times 256$
ReLU		-
H2S (MLP)	H2S (MLP)	$1 \times 32, 1 \times 32$
ReLU		-
$Startpoint_1$	$Startpoint_2$	$1 \times 2, 1 \times 2$
Input layer (MLP)	Input layer (MLP)	$1 \times 32, 1 \times 32$
ReLU		-
Concatenate	Concatenate	1×64[1] 1×64[1]
$Decoder_1(GRU)$	$Decoder_2(GRU)$	1×256[2] 1×256[2]
$Decoder_1(MLP)$	$Decoder_2(MLP)$	$1 \times 2, 1 \times 2$
$Ponit_1$	$Ponit_2$	$1 \times 2, 1 \times 2$
Collector[3]	Collector[3]	$50 \times 2, 50 \times 2$

[1] Here, the size input state of GRU is set to 64, and it comes from the Concatenation of start point (1×32) and H2S (1×32).

[2] The size of hidden state of GRU is set to 256, thus the input size should be 256 as well. This input state comes from the Hidden layer.

[3] This collector means the sequence is generated by a loop and each iteration outputs one point.

Then reparameterization trick is operated on z_{mean} and z_{std}. We use them to reparameterize a normal distribution to guarantee the possibility of back-propagation. The latent code z can be obtained after this operation. The architecture of our decoder is shown in Tab II. During the generating process, we firstly use a Multiple Layers Perception (MLP) to map the latent code z into an initial hidden state of GRU. The input of GRU consists of two parts and each of them has the dimension of 32. The first part comes from the hidden state through the *Hidden2Startpoint (H2S) Layer*, while the second part comes from the output point of last iteration. As for the first point of the sequence, an arbitrate start point can be used. In our experiment, we use $(0, 0)$ as the start point.

TABLE III

ARCHITECTURE OF ORIGINAL VAE

Description		Output dimension
$Sequence_1(x_1, y_1)$	$Sequence_2(x_1, y_1)$	$50 \times 2, 50 \times 2$
Concatenated sequence (x_1, y_1, x_2, y_2)		50×4
Input layer (MLP)		50×64
ReLU		-
Encoder (GRU)		1×256
Encoder (MLP)		256×20
z_{mean}	z_{std}	$1 \times 10, 1 \times 10$
z (Reparameterization)		1×10
Hidden layer (MLP)		1×256
ReLU		-
Start point (x_1, y_1, x_2, y_2)		1×4
Input layer (MLP)		1×64
ReLU		-
Decoder (GRU)		1×256
Decoder (MLP)		1×4
Two points		1×4
Collector[1]		50×4
$Sequence_1$	$Sequence_2$	$50 \times 2, 50 \times 2$

[1] This collector means the sequence is generated by a loop and each iteration outputs one point.

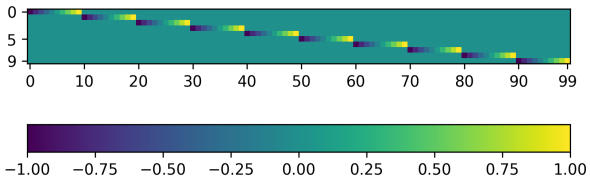

Fig. 1. **Test code used in experiments.** We display 100 test code vectors here and each vector contains 10 codes. The value of each elements is shown according to its color.

B. Original Variational Autoencoder

We build a VAE baseline for comparison. The main difference between VAE and our *MTG* is the way of using GRU. The architecture of encoder and decoder are displayed in Tab III. In this framework, we process the two sequence simultaneously and only use original GRU module.

C. InfoGAN

In this part, we display the infoGAN baseline used in our experiment. The whole architecture is displayed in Tab IV, where the discriminator and the Q module use the same front end. The two sequences are processed simultaneously as well. We set the dimension of the random noise to 32 and set the size of latent code to 10.

TABLE IV
ARCHITECTURE OF INFOGAN

Description		Output dimension
Random noise z	Latent code c	$1 \times 32, 1 \times 10$
Concatenated input		1×42
Generator (GRU)		1×256
MLP		1×128
ReLU		-
MLP		1×4
Tanh		-
Collector[1]		50×4
$Sequence_1$	$Sequence_2$	$50 \times 2, 50 \times 2$
Concatenated sequence (x_1, y_1, x_2, y_2)		50×4
Front-end (GRU)[2]		1×256
MLP		1×128
ReLU		-
Discriminator (MLP)	Q (MLP)	$1 \times 64, 1 \times 64$
Discriminator (MLP)	Q (MLP)	$1 \times 1, 1 \times 10$
Sigmoid	-	-
BCE Loss	Log Gaussian Loss	-

[1] This collector means the sequence is generated by a loop and each iteration outputs one point.
[2] This front-end is used in both discriminator and Q module, which can extract some common features.

II. TRAINING DETAILS

In this section, we will explain some hyper-parameters and training methods used in the training stage. We implement all of our experiment with *Pytorch*[1] and the hardware platform is $NVIDIA^{®}\ GTX1070\ (8G)$ and $Intel^{®}\ i7\text{-}7700K$. Values of all hyper-parameters are shown in Tab V. We use Adaptive Moment Estimation (Adam) algorithm [1] as our optimizer and the hyper-parameter *beta* of β-VAE is implemented with annealing method. The formula of the annealing process is the same as [2]:

$$\lambda_{step} = 1 - (1 - \lambda min) \times R_{step} \tag{1}$$

$$\beta = \beta_{base} \times \mu_{step} \tag{2}$$

$$\mathcal{L}_{VAE} = \mathcal{F}(S_1, \bar{S_1}) + \mathcal{F}(S_2, \bar{S_2}) + \beta \times KL(q||p) \tag{3}$$

The test codes that we use for sampling are described in Fig 1. Each vector only has one non-zero elements, and other elements are zero. The reason for this operation is to show the control ability of each code. The loss of VAE has two parts and they have their own tendency during the training process. In Fig 9, we display the values of two losses.

III. GENERATIVE RESULTS

A. More Generation Trajectories

In order to comprehensively display our system's performance, we will show more generative trajectory results, including using codes described in Fig 1 and using completely random codes. Fig 2 shows the results of test codes and Fig 3 shows those of random codes.

[1] https://pytorch.org/

TABLE V
HYPER-PARAMETERS DURING TRAINING STAGE

Parameter Name	Parameter Value
Batch size	400
Max iteration	10e5
z_{dim}	10
noise dimension (infoGAN)	32
Sequence length	50
learning rate (VAE)	0.0001
β_1 of Adam (β-VAE)	0.9
β_2 of Adam (β-VAE)	0.999
β_{base} in β-VAE	0.1
λmin in β-VAE	0.1
R_{step} in β-VAE	0.9995
learning rate (Discriminator in infoGAN)	0.0001
β_1 of Adam (Discriminator in infoGAN)	0.5
β_2 of Adam (Discriminator in infoGAN	0.99
learning rate (Generator in infoGAN)	0.0001
β_1 of Adam (Generator in infoGAN)	0.5
β_2 of Adam (Generator in infoGAN)	0.99

B. Zoom-in Display

For encounter trajectories, the distance between two data points reflects the speed of the vehicle. Therefore, we also display some zoom-in trajectories to show more details about the generative results. In these images, we an also check the index of the point to analyze the order of two sequences. Aforementioned results are shown in Fig 4.

Algorithm 1 New Disentangled Metric

Set $\Sigma = 0.1, 0.4, 0.7, 1.0, 1.3, 1.6, 1.9, 2.2, 2.5, 2.8$
Initiate Ω as an empty set
for each $i \in [1,\ z_{dim}]$ **do**
  // $\#\Sigma$ is the number of all variance values
  for each $\sigma \in [1,\ \#\Sigma]$ **do**
    // L *is number of samples for one variance*
    Initiate Θ as an empty set
    for each $l \in [1,\ L]$ **do**
      Generate one latent vector C from sampling c_i with $N(0, \sigma)$ and fixing others;
      // *through decoder of VAE*
      Latent code $C \implies$ One sample S;
      // *through encoder of VAE*
      One sample $S \implies$ Latent code $\hat{C}$;
      Append $\hat{C}$ to Θ;
    end for
    $\omega_{i,\sigma} = Var(\Theta)$
    Append $\omega_{i,\sigma}$ to Ω;
  end for
end for
Display variances in Ω;

IV. NEW DISENTANGLED METRIC

A. Detail Description

We provide the metric algorithm description here again and it can be found in Algorithm 1. Since there are three

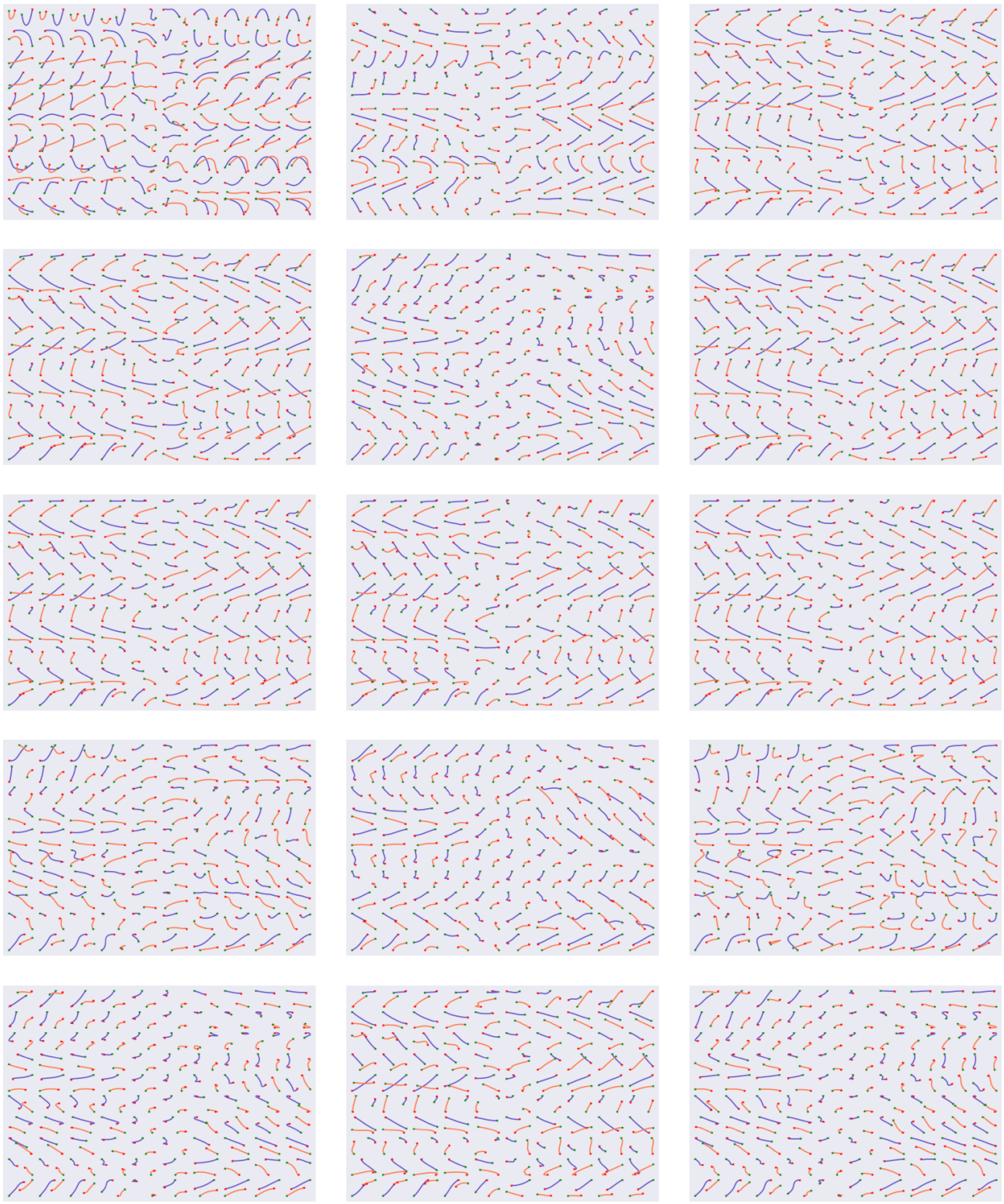

Fig. 2. **Generation results of MTG.** These results are generated by test codes. We display 12 figures here, each of which contains 100 sub-figures.

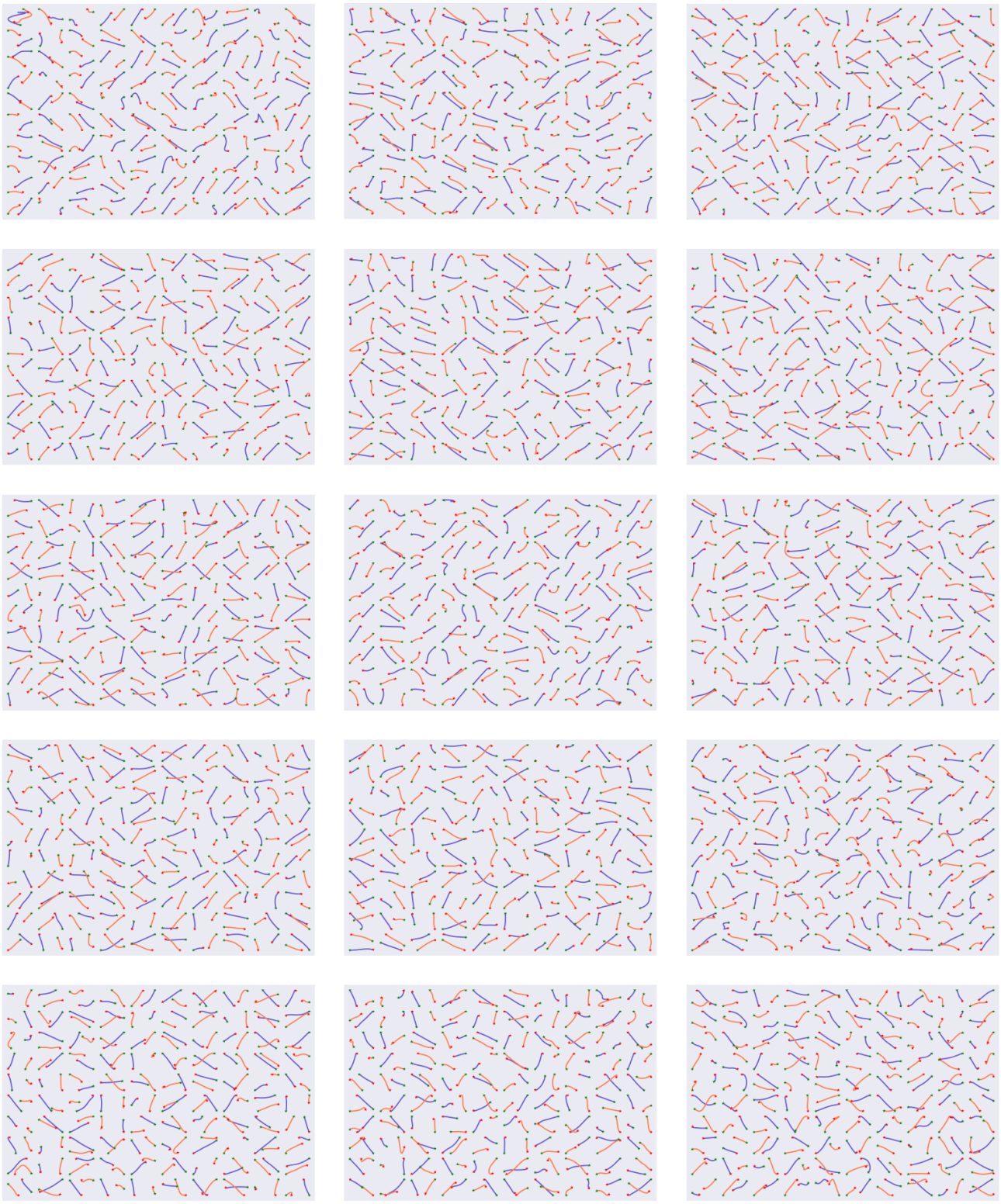

Fig. 3. **Generation results of MTG.** These results are generated by randomly selected codes. We display 12 figures here, each of which contains 100 sub-figures.

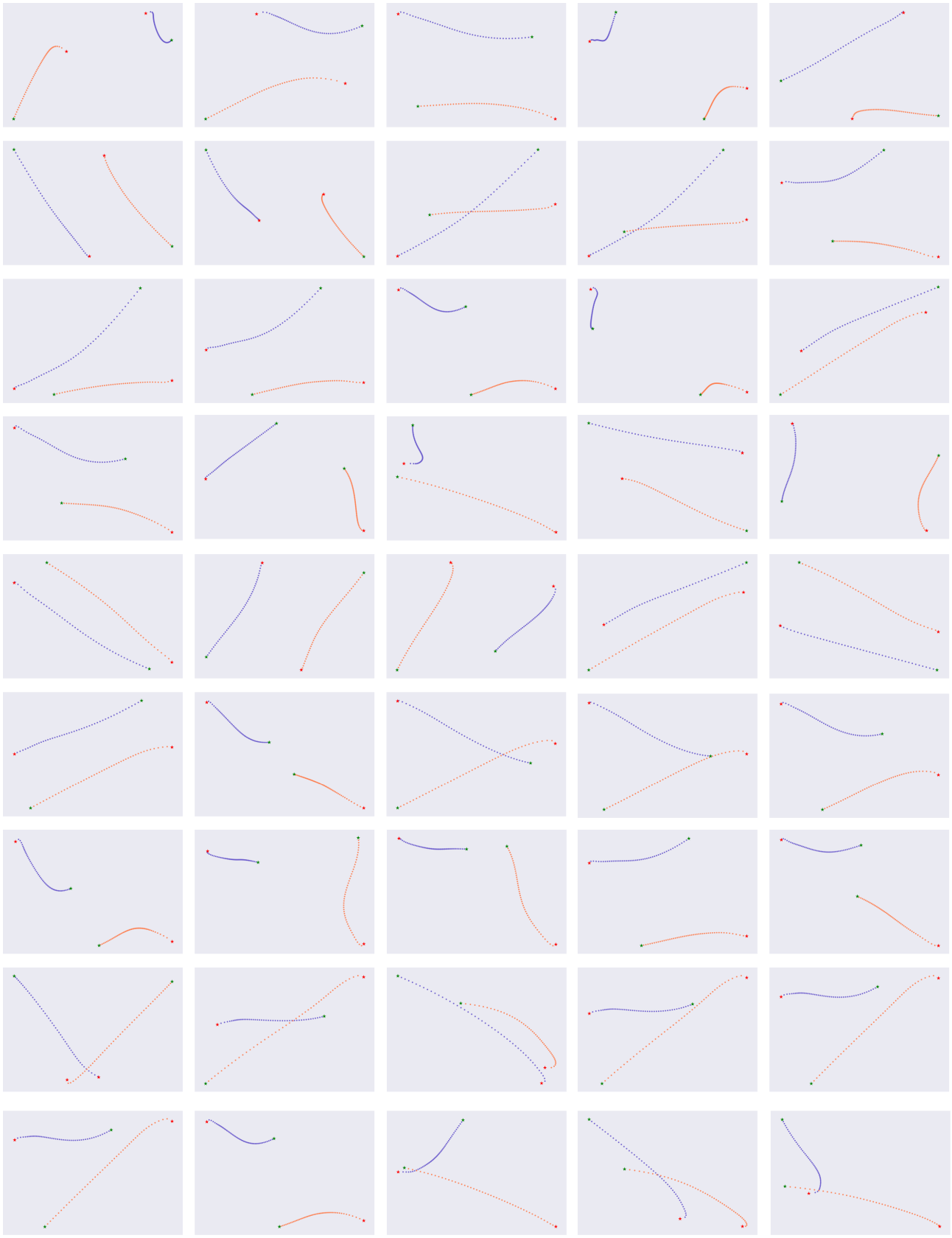

Fig. 4. **Zoom-in generation results of MTG.** These results are generated by test codes, and we draw them with scatter method. Each point represents a sampling point. Red point is the start point and green point is the end point.

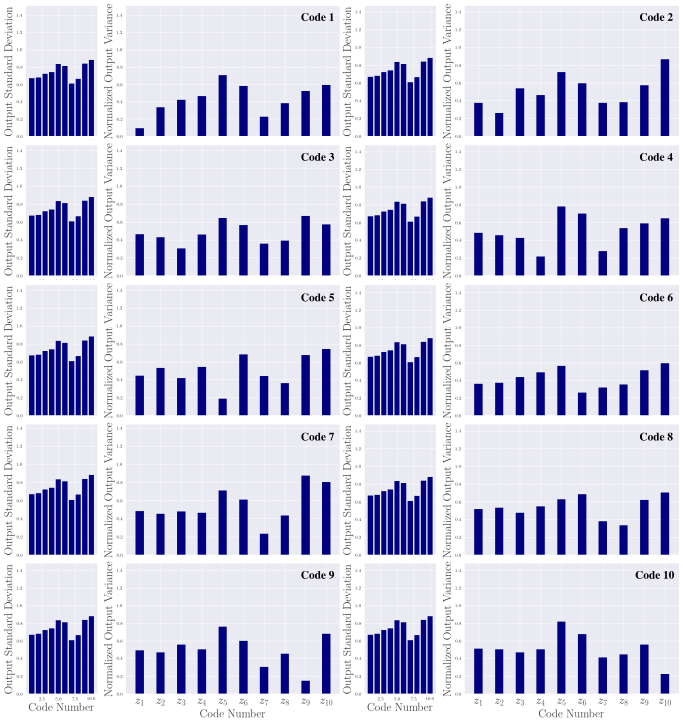

Fig. 5. **Results of Autoencoder tested on old metric.**

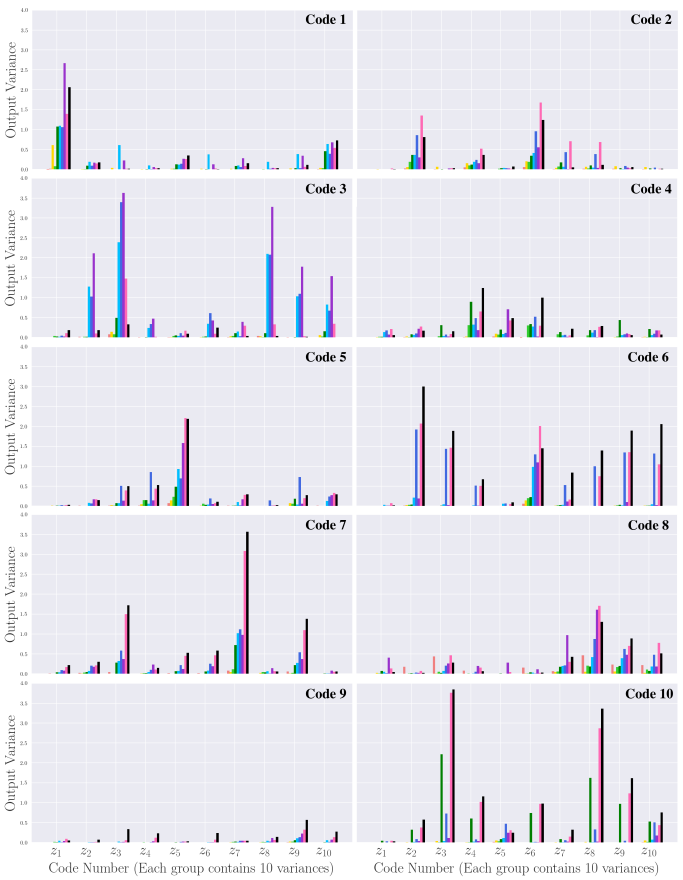

Fig. 6. **Results of Autoencoder tested on new metric.**

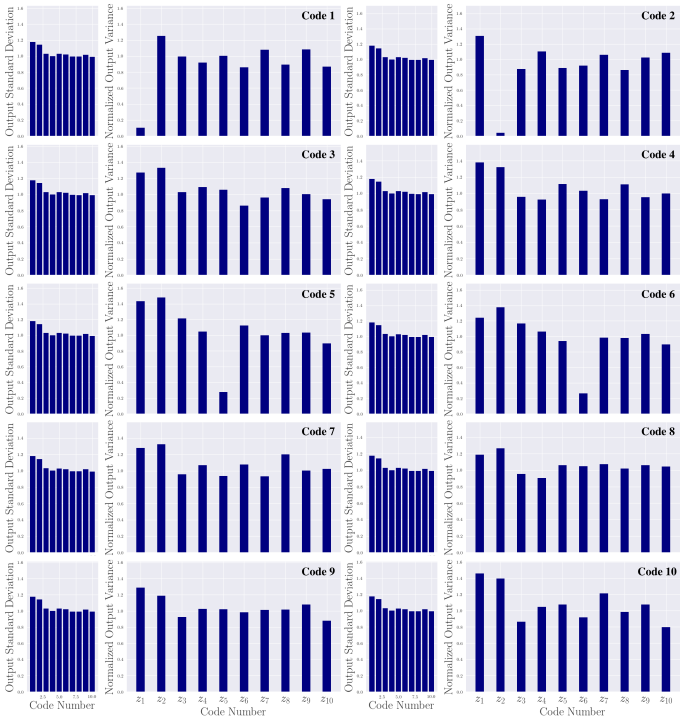

Fig. 7. **Results of VAE tested on old metric.**

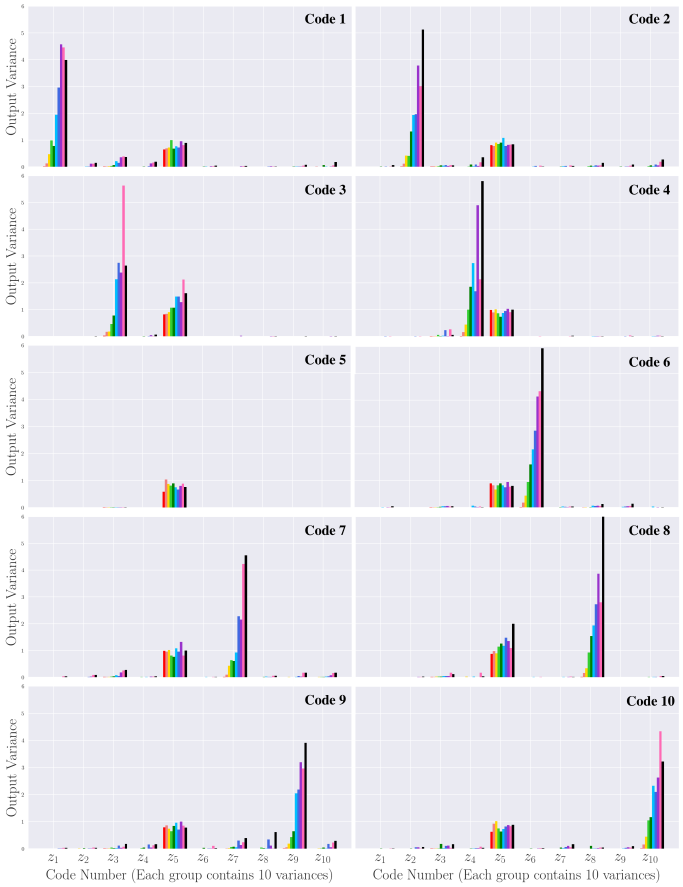

Fig. 8. **Results of VAE tested on new metric.**

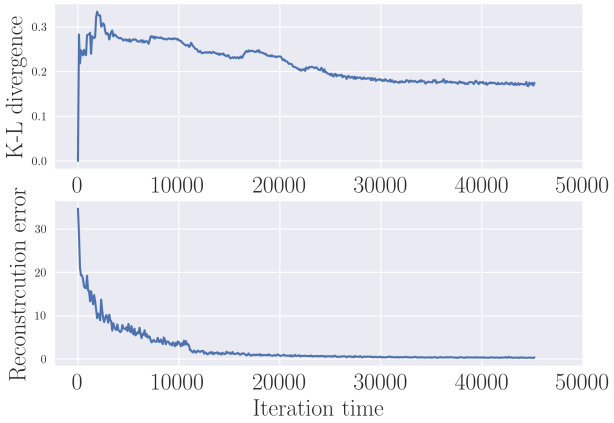

Fig. 9. **Loss value during training stage.** Two parts of the loss function are displayed separately. The top figure shows the value of K-L divergence, while the bottom figure shows the value of reconstruction error.

loops in the algorithm, the time complexity is $O(\sigma \times L \times N)$. Considering the calculating speed and accuracy, the number of L is set to 200 in our experiments. It is worthy to be noted that the larger L is, the more accurate the variance is.

B. Comparison with Old Metric

In order to compare our metric with the old one [3], we conduct these two metrics on both Autoencoder and VAE. Due to the limitation of space, we only show the code z_6 in our main text. Here we will display the results of all codes. The results are shown in Fig 5, 6, 7 and 8. Results of Autoencoder tested on old metric and new metric are shown in Fig 5 and 6 respectively. Results of VAE tested on old metric and new metric are shown in Fig 7 and 8 respectively.

REFERENCES

[1] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Computer Science*, 2014.

[2] D. Ha and D. Eck, "A neural representation of sketch drawings," *arXiv preprint arXiv:1704.03477*, 2017.

[3] H. Kim and A. Mnih, "Disentangling by factorising," *arXiv preprint arXiv:1802.05983*, 2018.