Generating Virtual Reality Stroke Gesture Data from Out-of-Distribution Desktop Stroke Gesture Data

Lin-Ping Yuan* HKUST
Boyu Li† HKUST (GZ)
Jindong Wang‡ Microsoft Research Asia
Huamin Qu § HKUST
Wei Zeng ¶ HKUST (GZ)

Abstract

This paper exploits ubiquitous desktop interaction data as an input source for generating virtual reality (VR) interaction data, which can benefit tasks like user behavior analysis and experience enhancement. Time-varying stroke gestures are selected as the primary focus because of their prevalence across various applications and their diverse patterns. The commonalities (e.g., features like velocity and curvature) between desktop and VR strokes allow the generation of additional dimensions (e.g., z vectors) in VR strokes. However, distribution shifts exist between different interaction environments (i.e., desktop *vs.*VR), and within the same interaction environment for different strokes by various users, making it challenging to build models capable of generalizing to unseen distributions. To address the challenges, we formulate the problem of generating VR strokes from desktop strokes as a conditional time series generation problem, aiming to learn representations that are capable of handling out-of-distribution data. We propose a novel architecture based on conditional generative adversarial networks, with the generator encompassing three steps: discretizing the output space, characterizing latent distributions, and learning conditional domain-invariant representations. We evaluate the effectiveness of our methods by comparing them with state-of-the-art time series generation models and conducting ablation studies. We further illustrate the applicability of the enriched VR datasets through two applications: VR stroke classification and stroke prediction.

Index Terms: Human-centered computing—Virtual reality

1 Introduction

Virtual reality (VR) interaction data, generated by users when using VR applications, play an important role in understanding user behaviors [8, 36] and experiences [17, 57]. The data can be analyzed and utilized from different perspectives to serve various stakeholders and purposes. For example, hand movement data can enable designers to offer stroke gestures as intuitive user interfaces in VR [32, 35]; gaze data associated with viewers' reactions to 360-degree videos can help storytellers refine their VR narratives [36, 38, 52]; embodiment interaction can help researchers understand space usage patterns [17]. However, the collection of VR interaction data faces various difficulties, such as a small user base due to limited VR device adoption [19], frequent deployment failures owing to device incompatibility [25], and inconvenient communication during tutorial sessions [7]. Consequently, current VR interaction datasets are primarily derived from a small number of participants in lab settings, resulting in limited sample sizes and diversity [19] and thus impeding the full exploitation of interaction data for various downstream applications [27, 37, 39, 48].

*e-mail: lyuanaa@connect.ust.hk
†e-mail: blibr@connect.hkust-gz.edu.cn
‡e-mail: jindong.wang@microsoft.com
§e-mail: huamin@ust.hk
¶e-mail: weizeng@hkust-gz.edu.cn (corresponding author)

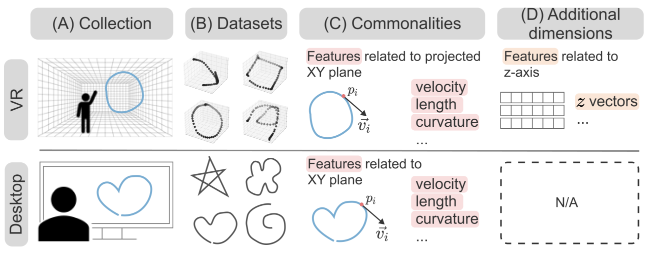

(A) Collection
(B) Datasets
(C) Commonalities
(D) Additional dimensions
VR
Desktop
Features related to projected XY plane
Features related to z-axis
p_i
$\vec{v_i}$
velocity
length
curvature
...
z vectors
Features related to XY plane
N/A

Figure 1: (A) Easy-to-collect desktop strokes may supplement VR stroke collection. (B) Desktop strokes add diversity to the VR stroke dataset with different shapes drawn by different users. (C) VR and desktops have commonalities, including features related to the XY plane such as velocity and curvature. (D) VR strokes present additional dimensions, such as z vectors.

To alleviate the difficulties in acquiring sufficient and diverse data, researchers have investigated various techniques [9, 29] to generate VR interaction data. For instance, deep learning models have been employed to generate viewers' scan paths for 360-degree images [29] and synthesize privacy-preserving eye-tracking VR datasets [9]. Nevertheless, these approaches still rely on existing VR interaction data as input and thus the diversity of the synthesized data is restricted by the input data's inherent diversity.

To address the above limitations, this study investigates whether and how interaction data collected from desktop environments can be used as an alternative source of input to enhance the quantity and diversity of synthesized VR interaction data. Desktop interaction data are considered because they can be collected from crowdsourcing platforms that are accessible to numerous users with easy setup. We choose stroke gestures as the primary experimental subject due to their widespread use across many VR and desktop applications [5, 20], compatibility with multiple input devices (e.g., hands, stylus pens, and mice) [8], and a wide range of geometrical variations [4, 8]. As shown in Fig. 1, compared to desktop strokes, VR strokes exhibit **additional dimensions** (e.g., z vectors) due to their 3D nature and the lack of physical surfaces [5]. Fortunately, VR and desktop strokes also share **commonalities** (e.g., features such as speed and curvature on the xy plane), allowing for potential transferability from desktop to VR. Besides, the rich diversity of desktop datasets (e.g., featuring various stroke shapes) can enhance the diversity within synthesized VR datasets.

To explore the research question of how desktop strokes can enrich VR stroke datasets while preserving the original characteristics, we first conduct preliminary studies by analyzing VR and desktop stroke datasets regarding the commonalities and additional dimensions. As to commonalities, we find that distribution shifts not only exist between desktop and VR stroke datasets but also intrinsically within each dataset (see Fig. 2). This poses the first challenge of ensuring that models trained on VR stroke datasets can generalize to desktop data that comes from unseen distributions (i.e., out-of-distribution). As to additional dimensions, we find that z vectors continuously spread out the output space (see Fig. 3), leading to the second challenge of capturing relationships between commonalities and additional dimensions from small real VR datasets.

We propose a two-stage method to enrich VR stroke datasets from desktop strokes and address the above two challenges. As shown in Fig. 4, the first stage utilizes the real VR datasets and learns the relationships between the commonalities and additional dimensions. The second stage then leverages the commonalities provided by desktop datasets as input. By applying the relationships learned in the first stage, the second stage generates the absent additional dimensions and thus synthesizes more diverse and larger VR stroke datasets. To tackle the first challenge, we regard strokes as time series data, because they exhibit temporal characteristics (e.g., timing, duration, and order of actions). Then, we formulate the problem of generating VR strokes based on desktop strokes as a conditional time series generation problem under out-of-distribution circumstances. Specifically, when training generative models on VR datasets, we characterize the latent domains that exist inside VR datasets and then learn domain-invariant representations to mitigate the distribution shifts among different latent domains. This enables generalization when confronting unseen distributions in desktop datasets. For the second challenge, we discretize the output space by performing clustering algorithms on additional dimensions (i.e., z vectors), reducing the burden of learning the relationships between the commonalities and additional dimensions. Bringing them all together, we propose a novel architecture based on conditional generative adversarial networks, whose generator wraps the above designs and consists of three steps (Fig. 5): discretizing the output space, characterizing latent domains, and learning conditional domain-invariant representations. To evaluate the effectiveness and usefulness of our methods, we first conduct quantitative comparisons with several state-of-the-art time series generative models [11, 31, 54] and perform ablation studies. Then, we create two applications that showcase the benefits of enriched VR datasets in classifying and predicting VR strokes.

In summary, our main contributions are twofold. First, to the best of our knowledge, we are the first to explore generating VR stroke gesture data from desktop stroke gesture data as an alternative input source that is out-of-distribution. We propose a time series generative network with novel designs of output space discretization and conditional domain-invariant representation learning. Second, we develop two applications that show the effectiveness and usefulness of the datasets enriched by our methods and demonstrate the potential opportunities opened by our methods. Our code and datasets are available at `https://github.com/yuanlinping/VRStrokeOOD`.

2 Related Work

2.1 Methods for Obtaining VR Interaction Datasets

There are mainly two ways to obtain VR interaction datasets. The first way is to collect from real users [22, 23, 36, 52]. Various sensors (e.g., controllers [8, 40, 55], motion trackers [23], and eye trackers [36, 52]) are attached to users, monitoring and recording their actions while they interact with virtual environments. These methods provide realistic data, but they can be time-consuming due to small user bases [19], deployment failures [12, 25], and tedious tutorials [7], leading to limited quantity and diversity.

The second way is to synthesize VR interaction data based on collected VR datasets [9, 29]. For example, Martin et al. [29] proposed a model to generate scan paths for 360-degree images with a spherical version of dynamic-time warping as loss functions. Brendan et al. [9] utilized conditional variational autoencoders to synthesize privacy-preserving eye-tracking VR datasets. Although these methods are efficient and scalable, they may not fully capture the variety of user interactions [53], because they still depend on the available VR datasets that are collected from small user groups.

In this study, we propose to use desktop interaction data, which can be collected from larger user groups with easier setup, as an alternative input source for generative models to increase the quantity and diversity of synthesized VR interaction datasets.

2.2 Stroke Gestures in VR and Desktop Environments

Stroke gestures refer to continuous hand or finger movements that convey user input through symbols or shapes drawn on a surface or in the air [20]. They play a critical role in many applications, including painting [47, 50], 3D modeling [27], and data analysis [24, 39]. This is because they offer users a natural and intuitive means of interaction by closely mimicking real-life actions and movements, enabling users to communicate their intuition and intention with systems with less learning curve and cognition load [21].

Researchers have studied stroke gestures from different perspectives to boost their potential. There are several shared topics between VR and desktop environments. For example, various algorithms are proposed to classify desktop (e.g., $P+ [42]) and VR (e.g., 3D Rubine [32]) stroke gestures. Studies have also examined how different input devices affect the drawing precision and speed to inform support for precise desktop [20, 41] and VR [8] stroke drawing. Besides, some studies have investigated issues unique to VR environments, such as reducing fatigue of mid-air drawing and improving draw performances by providing support [5, 51]. Conversely, research in desktop environments has delved into topics like stroke synthesis, such as how to enrich strokes produced by visually impaired people based on strokes produced by normal vision people [21].

However, no research has focused on synthesizing VR stroke gestures from desktop stroke gestures. It remains unknown whether and how the commonalities can serve as a bridge toward generating the additional dimensions. We propose a novel time series generative model that uses out-of-distribution techniques to solve the problem.

2.3 Out-of-distribution Generalization

Out-of-distribution (OOD) generalization [45] refers to the scenario that we have one or multiple training domains available and we want to learn a predictive function for unseen distributions. The key of OOD generation is to learn domain-invariant features, which can be achieved via different types of approaches [45]. Of all existing work in OOD generalization, two types are closely related to our work.

The first type is data augmentation or generation, which aims to get new training data by either harnessing existing augmentation techniques [46] or generating new samples using generative algorithms [44]. These approaches inspire us to view the enrichment of VR stroke datasets as a generative problem. However, they mostly focus on images and only consider spatial information, neglecting time series data where time relations play a critical role.

The second type is OOD generalization approaches for time series. One key difference between general OOD and time series OOD algorithms is that the domain labels are usually not given as a prior in time series data [10, 26]. However, the temporal covariate shifts exist in the time series data, which are often non-stationary and have changing statistical values. AdaRNN [10] and DIVERSIFY [26] are two recent OOD models for time series prediction and classification problems from a distribution perspective. Their core idea is to first split the entire time series into several domains by maximizing the distribution differences among the resulting domains, and then learn importance vectors to align hidden states of trained RNNs [10] or use a min-max adversarial game [26] to achieve domain distribution characterization and domain-invariant representation learning.

Our work is inspired by AdaRNN and DIVERSIFY for OOD generalization. However, their detailed network architectures cannot be simply applied to our problem because they focus on prediction and classification but generating VR stroke gestures is a generative problem. We propose a novel network architecture based on GANs, whose generator tailors the core idea of DIVERSIFY to generate VR stroke gestures with a novel label space discretization method and conditional invariant representation learning.

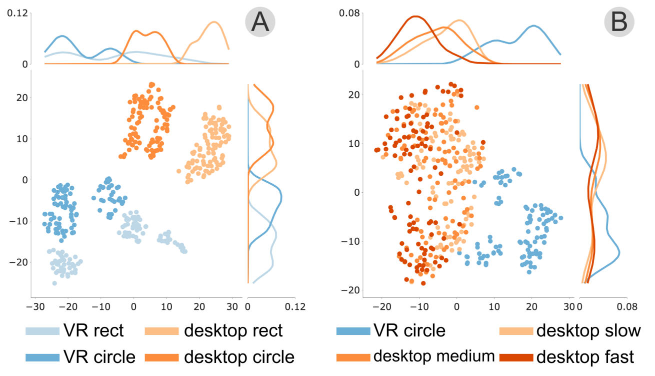


Figure 2: The KDE and t-SNE analyses on commonalities reveal distribution shifts *between* VR and desktop datasets, and *within* each.

3 Preliminary Studies

To inform the algorithm design, we first conduct preliminary studies by analyzing VR and desktop stroke datasets.

3.1 Data Preparation

To prepare data for analysis, we first preprocess raw stroke data, and then extract features based on preprocessed data. Later, when we train various generative models (Sec. 6 and Sec. 7), we also perform the same preprocessing and feature extraction to obtain input data.

Datasets. The preliminary studies use two datasets: (1) 3DMadLabSD VR stroke dataset [16], which consists of 40 stroke types drawn by 10 users, and (2) $1 desktop stroke dataset [49], which consists of 16 stroke types drawn by 10 users in slow, medium, and fast speeds. Each stroke in the 3DMadLabSD dataset is a list of (x,y,z,t) points, and each stroke in the $1 dataset is a list of (x,y,t) points, where x, y, z are coordinates and t represents a timestamp.

Data Preprocessing. With the raw datasets, we first perform data preprocessing. Since the number of points in each stroke varies in the raw data, we perform temporal resampling to standardize each stroke to 64 points. For each stroke, we divide its original time range into equal intervals and obtain a new list of points using cubic interpolation. Then, for each stroke, we normalize its x and y coordinates to $[0,1]$ and translate its z coordinate to start at 0.

Feature Extraction. With the preprocessed data, we perform feature extraction. Previous research [4, 41, 43] has characterized a stroke with various features, which can be categorized into two types: 1) geometric features that describe the appearance of a stroke, such as shape, size, and curvature, and 2) kinematic features that describe the movement during stroke production, such as speed and acceleration. Following their practices, we extract 15 features regarding the XY plane for each stroke in both 3DMadLabSD [16] and $1 dataset [49]. Specifically, we project VR strokes into the (x,y) plane, making the projected data comparable across the extracted features with the desktop data. A complete description and calculation of the extracted features can be found in the supplementary materials.

3.2 Data Analysis

After feature extraction, each desktop and VR stroke is described with a list of points, and each point is associated with (x,y) coordinates and features related to the XY plane. We regard the (x,y) and these features as commonalities between desktop and VR strokes. Each point in a VR stroke has a z coordinate, which is regarded as the additional dimension. Data analysis is performed on the commonalities and additional dimensions, respectively.

Data Analysis on Commonalities. We first group the strokes in the VR and desktop datasets based on their shapes (e.g., rectangles and circles) and then perform t-SNE and KDE analysis to investigate their distributions. Figure 2-A shows the comparison between VR rectangles, VR circles, desktop rectangles, and desktop circles. We

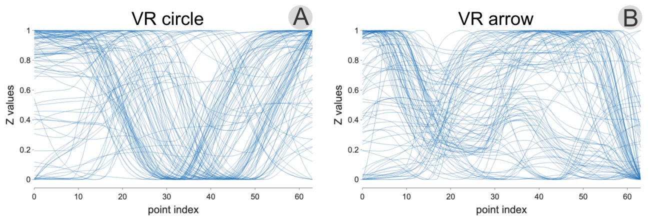


Figure 3: The additional z vectors of VR strokes spread out the entire output space and overlap between different stroke types. Each line represents a z vector of 64 dimensions.

can observe that VR and desktop strokes are well-separated in the t-SNE plot and their KDE histograms are different, indicating that distribution shifts exist *between* VR and desktop datasets. Even more intriguing is the observation that VR rectangles and VR circles are distinctively separated in the t-SNE plot and exhibit different KDE histograms. This indicates unexpected distribution shifts even *within* the VR datasets. A similar pattern also emerges for desktop rectangles and desktop circles, indicating distribution shifts *within* the desktop datasets. We further perform t-SNE and KDE analysis after grouping desktop strokes based on the drawing speeds. VR strokes are not grouped because users were not asked to draw at different speeds during collection [16]. Figure 2-B shows the comparison between VR circles and desktop circles drawn at slow, medium, and fast speeds. Again, we can observe that VR circles and desktop circles are well-separated, indicating the distribution shifts *between* VR and desktop data. For desktop circles at different speeds, their KDE histograms are different, indicating that drawing speeds can lead to distribution shifts *within* desktop data.

Data Analysis on Additional Dimensions. Figure 3 plots z vectors of two VR stroke types in the 3DMadLabSD dataset [16], with each line a z vector. When comparing the z vectors between VR circles and VR arrows, we find that the z vectors of different stroke types exhibit overlapping patterns that closely mirror each other. This suggests shared characteristics in the manner users draw these strokes. When observing each stroke type individually, we find that the z vectors also exhibit considerable variation. This variability could stem from the depth inaccuracies or deviations when drawing in VR, where users tend to draw on a curved surface despite their efforts to restrict their hand movements on an imaginary flat surface, as noted in previous research [5]. We also observe that some z vectors depict an initial high value tapering off towards the end, while others show the opposite trend or even peak in the middle. This might be because the starting point and moving arcs of users' arms [5] and wrists [34] influence the deviations. For example, when drawing a circle, users might start at the top and then reach the bottom, or start at another point with different moving arcs. Besides, we notice a greater variation in the z vectors of VR arrows compared to VR circles. This is likely attributed to the sharp turns in drawing arrows, which involve more complex hand movements and thus lead to more depth deviations. This observed variability indicates a continuous distribution of z vectors within the output space.

3.3 Findings

We summarize the findings as follows, which inform us of the challenges we need to address when designing effective algorithms for generating VR stroke gestures based on desktop stroke gestures.

- For common parts between VR and desktop datasets, distribution shifts exist not only *between* VR and desktop datasets but also *within* VR datasets or desktop datasets. The distribution shifts can be caused by multiple factors, such as input environments (i.e., VR or desktop), stroke shapes, drawing speeds, and other unknown reasons. The distribution shifts necessitate that models trained on VR strokes possess robust generalization capabilities to perform well on out-of-distribution desktop data.

- For additional dimensions of VR datasets, the z vectors exhibit great variability. They take up the output space continuously and overlap across different stroke types, making it hard for models to capture relationships between commonalities and additional dimensions from small real VR datasets.

4 Problem Formulation

Commonalities and Additional Dimensions. Suppose two separate stroke gesture datasets have been collected from VR and desktop environments, respectively. The VR dataset is denoted by $V = \{V_1, V_2, \dots, V_n\}$ and the desktop dataset by $W = \{W_1, W_2, \dots, W_m\}$, where $m \gg n$. Each stroke gesture $V_i \in \mathbb{R}^{p \times q}$, where p denotes the number of points in the stroke and q encompasses the features of each point. Specifically, these features include the x, y, z coordinates, features related to the XY plane, denoted as $\mathbf{f}^{xy}$, and features associated with the Z-axis, denoted as $\mathbf{f}^{z}$. Similarly, a stroke gesture $W_j \in \mathbb{R}^{p \times r}$. Here, p represents the number of points, and r represents features consisting of the x, y coordinates and the XY plane features $\mathbf{f}^{xy}$. Thus, the commonalities between V_i and W_j are $\{x, y, \mathbf{f}^{xy}\}$, and VR additional dimensions are $\{z, \mathbf{f}^{z}\}$.

Latent Domains in V and W. The preliminary studies show that the values of $\{x, y, \mathbf{f}^{xy}\}$ in both V and W may draw from different distributions (Fig. 2). Since the distribution shifts are caused by various known and unknown factors, such as different input devices, shapes of stroke gestures, and drawing speeds, it is improper to simply divide them into several domains based on any single factor [26]. Instead, we propose to treat them as compositions of several (unknown) latent distributions to model their relationships.

Problem Formulation. Based on the above descriptions, we formulate our problem of generating an enriched VR stroke dataset V_{new} based on the real VR dataset V and desktop dataset W as a conditional generation task in the out-of-distribution settings [45].

Specifically, suppose that V consists of k latent domains $\mathcal{D}_{train} = \{\mathcal{D}_i | i = 1, \dots, k\}$. $\mathcal{D}_i = \{\mathcal{A}_i, \mathcal{C}_i, P_i(\mathbf{a}, \mathbf{c})\}$ describes the i-th domain, where $\mathcal{A}$ is the value space of additional dimensions (i.e., $\{z, \mathbf{f}^{z}\}$) that will be generated, $\mathcal{C}$ is the value space of commonalities (i.e., $\{x, y, \mathbf{f}^{xy}\}$) on which generation is conditioned, and $P(\mathbf{a}, \mathbf{c})$ is the joint probability distribution. The joint distributions between each pair of latent domains differ: $P_i \neq P_j$ for $1 \leq i \neq j \leq k$.

A conditional generative model $G : \mathcal{C} \times \mathcal{Z} \rightarrow \mathcal{A}$, where $\mathcal{Z}$ represents the latent space, can be built to generate instance $a \in \mathcal{A}$ conditioned on $c \in \mathcal{C}$, namely generating values of additional dimensions based on values of common parts. In our problem, we aim to learn a generalized G using the k training domains from the real VR dataset V that can generate values for additional dimensions based on desktop dataset W, which is not accessible during training and whose common parts are drawn from unseen distributions.

The generated values of additional dimensions should preserve the characteristics shown in the real VR dataset V to reflect common or general habits of how users produce VR strokes. Mathematically, given a specific c in C, the marginal distribution of the generated values based on c, denoted as $P_{G|c}$, should be closely aligned with the marginal distribution of the values in V corresponding to that same c, denoted as $P_{V|c}$. The objective can be formulated as:

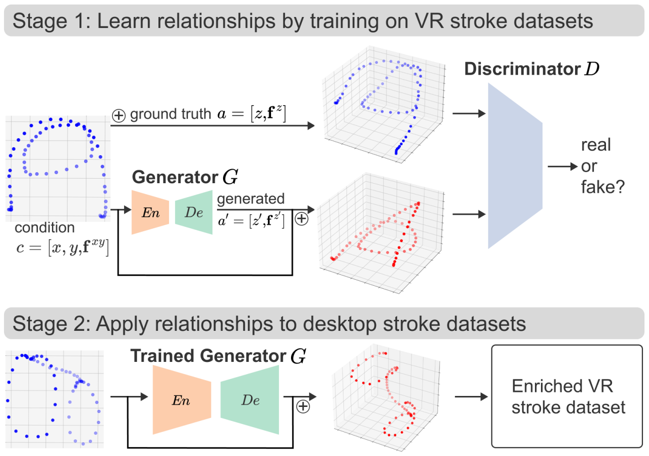
$$\min_{G} D(P_{G|c}, P_{V|c}), \tag{1}$$

where $D(\cdot, \cdot)$ represents a suitable distance measure.

5 Proposed Methods

5.1 Method Overview

To increase the diversity of generated VR stroke gestures, we propose to take desktop stroke gestures as an alternative input. To achieve this, we regard stroke gestures as time series data and propose a *conditional time series generative model*. As illustrated in Fig. 4, our method consists of two stages:



Stage 1: Learn relationships by training on VR stroke datasets
⊕ ground truth $a = [z, \mathbf{f}^z]$
Discriminator D
real or fake?
Generator G
En
De
generated $a' = [z', \mathbf{f}^z]$
condition $c = [x, y, \mathbf{f}^{xy}]$
Stage 2: Apply relationships to desktop stroke datasets
Trained Generator G
En
De
Enriched VR stroke dataset

Figure 4: Our two-stage method learns and applies the relationships between the commonalities and additional dimensions.

Stage 1: learn relationships by training on VR stroke datasets. We propose a novel network architecture based on conditional generative adversarial networks (cGAN) [18]. Specifically, the network consists of a generator G and a discriminator D. Given a VR stroke gesture in the real VR stroke dataset, the generator incorporates its $[x, y, \mathbf{f}^{xy}]$ features (i.e., commonalities) as a conditional input, and generates the z vector (i.e., additional dimensions) conditioning on $[x, y, \mathbf{f}^{xy}]$. After concatenating the conditional input $[x, y, \mathbf{f}^{xy}]$ with the generated z vector, we get a complete generated VR stroke. The discriminator then judges whether the generated stroke conforms to the distribution of real VR strokes (Eq. 1).

Stage 2: apply relationships to desktop stroke datasets. After training a cGAN model on the real VR stroke dataset, the second stage is to use the trained generator G and desktop strokes to generate more VR strokes. Specifically, the trained generator G generates z vectors conditioning on $[x, y, \mathbf{f}^{xy}]$ provided by a desktop stroke and preserves the characteristics of the real VR datasets by applying the learned relationships. Since desktop strokes can offer more diverse shapes and geometry variants, the generated VR stroke datasets will also contain the shapes and geometry variants that are not presented in the real VR stroke datasets.

Although some cGANs specific for time series [11, 31, 54] could be candidates for the first stage, they may be insufficient to address distribution shifts and the continuous output space reported in Sec. 3.3. Therefore, we further propose our solution as a *conditional time series generative model under out-of-distribution circumstances*, and modify the generator in cGAN [18] to enable the generator to handle the distribution shift and continuous output space issues. Next, we will elaborate on our generator design.

5.2 Conditional Domain-Invariant Generator

To make our generator generalizable to unseen distributions, we borrow and adapt the core ideas of DIVERSIFY [26].

DIVERSIFY [26] inspects time series classification from a distribution perspective. Specifically, since time series data are often non-stationary, DIVERSIFY [26] argues that a time series does not follow one fixed distribution, but has several latent distributions, forming several latent domains. However, time series data often do not have proper domain labels, necessitating to characterization of the latent distributions and domains with advanced algorithms. To classify time series that belong to several latent domains while domain labels are missing, DIVERSIFY [26] proposes three steps: (1) *fine-grained feature update* that utilizes pseudo domain-class labels to extract features, (2) *latent distribution characterization* that divides the whole time series dataset into several latent domains, and (3) *domain-invariant representation learning* that enables generaliza-

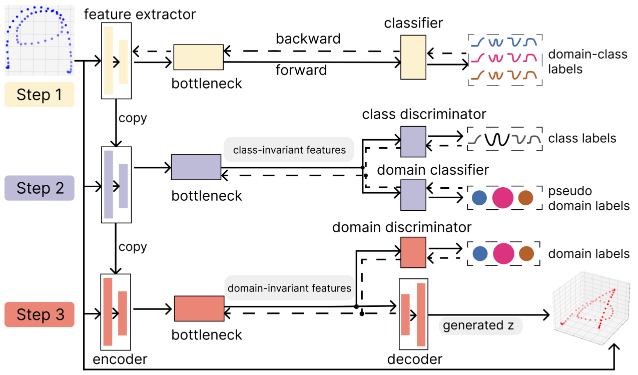


Figure 5: The proposed generator (G in Fig. 4) has three steps: discretizing the output space, characterizing latent domains, and learning conditional domain-invariant representations.

tion across different domains. In the three steps, DIVERSIFY [26] maximizes the distribution differences among the latent domains and learns domain-invariant representations that can bridge the maximal distribution gaps to achieve the best generalizability.

As stated in Sec. 3.3 and Sec. 4, both VR and desktop stroke datasets have distribution shifts, indicating the existence of latent domains. Moreover, since distribution shifts can be caused by several factors, such as shapes and drawing speeds, it is hard to simply label the domains with a single factor, and thus we borrow the second step of DIVERSIFY to characterize the latent domains. However, DIVERSIFY is for time series classification with a limited set of possible outputs (i.e., class labels) but our problem is a generative problem with a continuous output space. Thus, we modify the first and third steps of DIVERSIFY and propose a conditional domain-invariant generator with three steps (Fig. 5): (1) *discretizing the output space* to characterize possible z vectors, (2) *characterizing latent distributions* to divide the real VR datasets to several latent domains, and (3) *learning conditional domain-invariant representations* that are generalizable to generate z vectors conditioning on $[x, y, \mathbf{f}^{xy}]$ that come from unseen distributions.

5.2.1 Discretizing Output Space

As reported in Sec. 3.3, z vectors can spread out the entire output space, and the correspondence between features in common parts (i.e., $[x, y, \mathbf{f}^{xy}]$) and additional dimensions (i.e., z vectors) intertwine together, making it hard for generators to capture their relationships. To reduce the difficulties, we first apply the k-nearest neighbor clustering algorithm to cluster the z vectors of all strokes in the real VR dataset. Figure 6 shows an example of the ten clusters obtained from the 3DMadLabSD dataset [16]. Then, we utilize the resulting clusters to discretize the output space by treating them as ten classes and assigning each z vector a corresponding class label l_z.

VR strokes with the same class label may come from different latent domains. We train a feature extractor, a bottleneck, and a classifier by utilizing pseudo domain-class labels [26] (Fig. 5-Step 1). Specifically, assuming that there are K latent domains and J classes/clusters, resulting in a total of $K \times J$ domain-class labels. The pseudo domain-class label $s \in \{1, 2, \cdots, K \times J\}$ for each stroke will be obtained and updated in each training iteration at the second step later. This step takes a VR stroke's commonalities $\mathbf{c}$ (i.e., $\{x, y, \mathbf{f}^z\}$) as input and outputs a domain-class label. The process is supervised learning and the loss function uses cross-entropy ℓ:

$$\mathcal{L}_{super} = \mathbb{E}_{(\mathbf{c},\mathbf{a})\sim\mathbb{P}^{tr}} \ell(s', s), \tag{2}$$
$$s' = C^{(1)}(B^{(1)}(F^{(1)}(\mathbf{c}))),$$

where $C^{(1)}$, $B^{(1)}$, and $F^{(1)}$ represent the classifier, bottleneck, and feature extractor at the first step, respectively. In the following, we use the superscript to denote the order of steps.

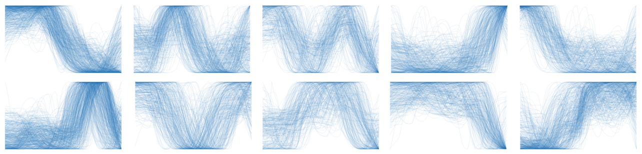


Figure 6: Examples of discretizing output space by clustering z vectors of 3DMadLabSD dataset [16].

5.2.2 Characterizing Latent Distributions

Following the practices in DIVERSIFY [26], the second step is to characterize the latent distributions and obtain latent domain labels for each VR stroke, rather than simply dividing the strokes into domains by any single factor (e.g., shapes or speeds). First, we use a class discriminator $Adv^{(2)}$ to classify classes (i.e., different z vector clusters) and learn class-invariant features $\mathbf{f}_{cls}$. We also use a domain classifier to obtain temporary domain labels l_d' (Fig. 5-Step 2). We iteratively perform k-means clustering algorithms on the class-invariant features to obtain K latent domains. The centroids of the domains are computed as:

$$\tilde{\mu}_k = \frac{\sum_{\mathbf{c}_i \in \mathcal{C}^{tr}} \delta_k l_d' \mathbf{f}_{cls}}{\sum_{\mathbf{c}_i \in \mathcal{C}^{tr}} \delta_k l_d'} \tag{3}$$

$$l_d' = (C^{(2)}(B^{(2)}(F^{(2)}(\mathbf{c}_i)))), \qquad \mathbf{f}_{cls} = B^{(2)}(F^{(2)}(\mathbf{c}_i)),$$

where δ_k is the k^{th} element of the logit soft-max output and $C^{(2)}$, $B^{(2)}$, and $F^{(2)}$ represent the classifier, bottleneck, and feature extractor at the second step, respectively.

Second, we calculate the distance between each class-invariant feature vector $\mathbf{f}_{cls}$ and the centroid, and assign the feature to the domain with the closest centroid:

$$\tilde{l}_d^i = \arg\min_k Dist(\mathbf{f}_{cls}, \tilde{\mu}_k). \tag{4}$$

Then, we can re-calculate the centroids and assign domain labels based on the distance to different centroids:

$$\mu_k = \frac{\sum_{\mathbf{c}_i \in \mathcal{C}^{tr}} \mathbb{I}(\tilde{l}_d^i = k)\mathbf{f}_{cls}}{\sum_{\mathbf{c}_i \in \mathcal{C}^{tr}} \mathbb{I}(\tilde{l}_d^i = k)}, \tag{5}$$
$$l_d^i = \arg\min_k Dist(\mathbf{f}_{cls}, \mu_k),$$

Here, $\mathbb{I}(\tilde{l}_d^i = k)$is 1 when $\tilde{l}_d^i = k$, otherwise it is 0.

The loss function for this step is composed of the class discriminator for adversarial learning and the domain classifier.

$$\mathcal{L}_{latent} = \mathbb{E}_{(\mathbf{c},\mathbf{a})\sim\mathbb{P}^{tr}} \ell(l_d', l_d) + \ell(Adv^{(2)}(B^{(2)}(F^{(2)}((c)))), l_z), \tag{6}$$

5.2.3 Learning Conditional Domain-Invariant Representations

Generating VR strokes from desktop strokes is a time series generation problem. The original DIVERSIFY [26] is for time series classification. Thus, we embed an encoder-decoder structure, which generates z vectors conditioning on $[x, y, \mathbf{f}^{xy}]$ of a stroke. The output of the whole generator can be represented as:

$$\mathbf{a} = De^{(3)}(B^{(3)}(En^{(3)}(\mathbf{c}))), \tag{7}$$

where $\mathbf{a}$ is $\{z, \mathbf{f}_z\}$ (z vector in our work since we only generate z this time), $De^{(3)}$, $B^{(3)}$, and $En^{(3)}$ are the decoder, bottleneck, and encoder (Fig. 5-Step 3), respectively.

After obtaining the latent domain labels for each VR stroke in the second step, this step aims to learn domain-invariant representations and enable the trained encoder-decoder can generalize to unseen distributions. To achieve this, we first use the feature extractor trained in the first step as the encoder, which captures knowledge in both domains and classes. Then, we utilize adversarial learning

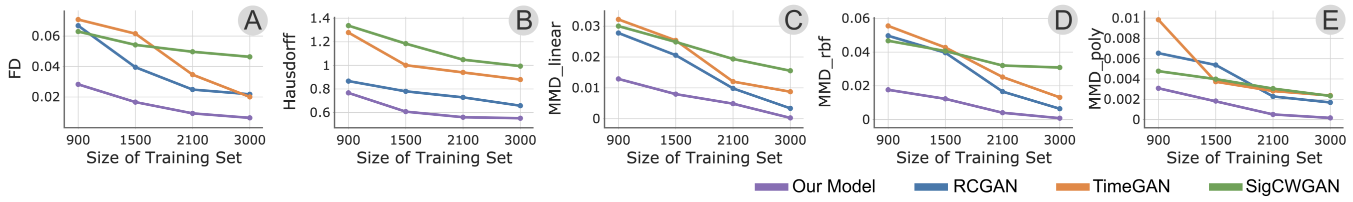


Figure 7: Comparison of our model with three baseline models RCGAN [11], TimeGAN [54], and SigCWGAN [31] trained with varying numbers of VR strokes. Lower metric values indicate better performance.

Table 1: Comparison of our model with three baseline models RCGAN [11], TimeGAN [54], and SigCWGAN [31] trained with 3000 VR strokes. Lower metric values indicate better performance. Boldface highlights the best results.

	FD	Hausdorff	MMD_linear	MMD_rbf	MMD_poly
RCGAN	0.021814	0.657808	0.003368	0.006400	0.001685
TimeGAN	0.020063	0.879646	0.008755	0.013145	0.002367
SigCWGAN	0.046372	0.993806	0.015546	0.030862	0.002329
Our Model	**0.006323**	**0.551768**	**0.000272**	**0.000799**	**0.000160**

to fool a domain discriminator $Adv^{(3)}$ that classifies domains. If the discriminator cannot differentiate between features of strokes from different latent domains extracted by the encoder, it indicates that the encoder has successfully learned domain-invariant features and bridged the distribution shifts among latent domains [13]. The domain-invariant features capture characteristics of how users draw VR strokes regardless of latent domains and thus allow decoders to generate z vectors that preserve the characteristics.

The loss function of the domain discriminator $Adv^{(3)}$ is:

$$\mathcal{L}_{domain} = \mathbb{E}_{(\mathbf{c},\mathbf{a})\sim\mathbb{P}^{tr}}\ell(Adv^{(3)}(B^{(3)}(En^{(3)}(\mathbf{c})),l_d). \tag{8}$$

5.3 Training Strategy

The training strategy of this network is relatively intricate. With the loss functions Eq. 2 and Eq. 6, we first train the feature extractor $F^{(1)}$ (becoming encoder $En^{(3)}$ later) in the first step and then train the class discriminator $Adv^{(2)}$ and domain classifier $C^{(2)}$ in the second step. Before proceeding with the third step in the generator, we train the discriminator D (Fig. 4) because the decoder $De^{(3)}$ in the third step is employed to deceive the discriminator D. To achieve this, we first freeze the parameters in the generator G and update the weights of the discriminator D. The employed loss function is as follows:

$$\begin{aligned}\mathcal{L}_D(G,D) =&\mathbb{E}_{(\mathbf{c},\mathbf{a})\sim\mathbb{P}^{tr}}[logD(\mathbf{c},\mathbf{a})]+ \\ &\mathbb{E}_{(\mathbf{c},\mathbf{a})\sim\mathbb{P}^{tr}}[log(1-D(\mathbf{c},G(\mathbf{c})))]\end{aligned} \tag{9}$$

We then go back to the third step of the generator G. We freeze the model parameters of the discriminator D and use the trained discriminator D to assess the results generated by the generator, thereby obtaining the discriminator loss for the generator:

$$\mathcal{L}_G(G,D) = \mathbb{E}_{(\mathbf{c},\mathbf{a})\sim\mathbb{P}^{tr}}[log(1-D(\mathbf{c},G(\mathbf{c})))]. \tag{10}$$

In addition, research [33] has indicated that incorporating an L2 loss into the generator's loss function can enable the generator to produce results that closely resemble the target:

$$\mathcal{L}_{L2}(G) = \mathbb{E}_{(\mathbf{c},\mathbf{a})\sim\mathbb{P}^{tr}}[(\mathbf{a}-G(\mathbf{c}))^2]. \tag{11}$$

Together with the loss of domain discriminator $Adv^{(3)}$, the final loss function for the third step in the generator G is as follows:

$$\mathcal{L} = \mathcal{L}_G(G,D) + \mathcal{L}_{L2}(G) + \mathcal{L}_{domain}. \tag{12}$$

6 Experiments and Evaluation

The section evaluates the effectiveness of the proposed method based on real VR stroke datasets (Stage 1 in Fig. 4). We use the 3DMadLabSD VR dataset [16] in our experiments. Ten users drew 40 types of strokes ten times each, yielding a total of 4000 VR strokes in this dataset. The 40 stroke types fall into four categories: ten Arabic numerals (i.e., 0-9), ten English letters (i.e., a-j), ten basic primitives (e.g., rectangles, circles, and arrows), and ten relatively complex symbols (e.g., combinations of circles, triangles, and lines). We perform data preprocessing and feature extraction based on these raw VR strokes by following the steps mentioned in Sec. 3.1.

6.1 Evaluation Metrics

We assess whether the VR strokes generated from 2D strokes preserve the characteristics of the real drawing trajectory of users in VR environments. To achieve this, we use several common metrics for GAN evaluation based on previous studies [29, 31, 54]. These metrics include (1) Fréchet distance (FD), (2) Hausdorff distance, and (3) Maximum mean discrepancy (MMD). The smaller these metrics, the better the performance of the time series generative model. Specifically, Fréchet distance measures the distance between two multivariate normal distributions and is commonly used to evaluate the similarity between two datasets. Hausdorff distance calculates the maximum distance between any point in one dataset and its nearest point in the other dataset, which is often used to measure the dissimilarity between two sets of points. Maximum mean discrepancy measures the distance between two probability distributions. We utilize three kernel functions in MMD computation: the linear, RBF, and polynomial kernel.

6.2 Comparison to Baselines

Baselines. We consider three baseline models: RCGAN [11], TimeGAN [54], and SigCWGAN [31]. They are all based on GANs and tailored for time series generation. RCGAN [11] merges recurrent neural networks with a conditional GAN framework. TimeGAN [54] further refines this approach by incorporating a time-series embedding to capture temporal dynamics effectively. SigCWGAN [31] introduces signature transforms as a novel technique to model complex sequential properties. These models share a common assumption: the training and testing datasets are drawn from the same distribution. In contrast, our model is specifically designed to deal with out-of-distribution samples, integrating mechanisms to characterize latent distributions and learn domain-invariant representations, which are absent in the baseline models.

Experiments. We aim to investigate two aspects. The first is to assess the generalizability of our proposed model against the three baselines. The second is to examine the influence of training data size on model performance. To achieve these, we partition the 3DMadLabSD dataset [16] into training and testing data based on stroke shapes. We ensure that shapes present in the testing data are

Table 2: Abalation studies on output space discretization.

	random class labels	z vector cluster labels
FD	0.030495	**0.006323**
hausdorff	0.667595	**0.551768**
mmd_linear	0.010434	**0.000272**
mmd_rbf	0.020506	**0.000799**
mmd_poly	0.001086	**0.000160**

Table 3: Abalation studies on the loss functions of the generator G.

	$\mathcal{L}_G(G,D)$	$\mathcal{L}_{L2}(G)$	$\mathcal{L}_G + \mathcal{L}_{L2}$
FD	0.134043	0.013179	**0.006323**
hausdorff	1.121143	0.559428	**0.551768**
mmd_linear	0.071677	**0.000008**	0.000272
mmd_rbf	0.129512	0.001704	**0.000799**
mmd_poly	0.006074	0.000285	**0.000160**

excluded from the training data, thereby guaranteeing different distributions necessary for assessing generalizability. Specifically, the training data comprises strokes from English letters, basic primitives, and relatively complex symbols, with 30 stroke types and 100 instances each, totaling 3000 strokes. The testing data includes strokes from Arabic numerals, with 10 stroke types and 100 instances each, totaling 1000 strokes. To investigate the impact of training set size, we construct three subsets from the training data by randomly sampling 30, 50, and 70 instances from each stroke type, resulting in subsets containing 900, 1500, and 2100 strokes, respectively. We proceed to train our model and the baselines on these four training sets with varying sizes (i.e., 900, 1500, 2100, and 3000 strokes) and assess their performance using the same testing data of 1000 strokes.

Result analysis. Figure 7 illustrates the relationship between the size of the training sets and model performance. Table 1 presents the numerical results for the models trained with 3000 VR strokes. Our model consistently outperforms the baselines across all metrics and data sizes. This suggests that our model has a superior ability to generalize when confronted with out-of-distribution testing data, synthesizing strokes that are closer to the target distributions. Furthermore, a positive correlation between data size and model performance is observed across all models, and our model exhibits better robustness. The findings suggest that, while increasing dataset size generally improves performance, our model requires less data to learn distributions than baselines, offering an advantage in scenarios where large VR datasets are not readily available.

6.3 Ablation Studies

We conduct ablation experiments to evaluate the impact of the output space discretization and the loss functions. All the following conditions take the 3000 strokes as training sets and the 1000 non-overlapping strokes as testing sets, the same as Sec. 6.2.

Output space discretization. As detailed in Sec. 5.2.1, we discretize the output space by clustering z vectors and assigning each vector a class label. To evaluate it, we perform a comparison experiment where we assign random class labels to VR strokes without considering their z vectors. As shown in Table 2 and Fig. 8-AB, the results reveal that the standard model trained with class labels based on z vector clusters outperforms the one with random class labels. This suggests that incorporating the z vector clusters can aid in characterizing latent domains and capturing relationships between commonalities and additional dimensions.

Loss functions of the generator. As shown in Eq. 12, the loss function of generator G contains a discriminator loss $\mathcal{L}_G(G,D)$ and a L2 loss $\mathcal{L}_{L2}(G)$. We evaluate the impact of the two components in aiding the model to learn the distributional characteristics of the latent domains by removing each of them from the generator G's loss function. Table 3 and Fig. 8-ACD present the results of models trained with $\mathcal{L}_G(G,D)+\mathcal{L}_{L2}(G)$, only $\mathcal{L}_G(G,D)$ and only $\mathcal{L}_{L2}(G)$. It is evident that the model's performance noticeably decreases when the discriminator loss or L2 loss is removed. This highlights the crucial role of the combined discriminator loss and L2 loss in training the generator.

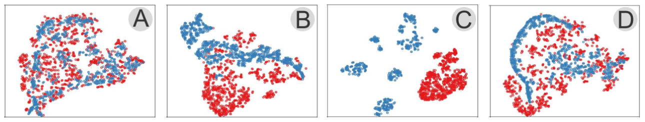


Figure 8: The t-SNE plots of testing results of models trained on different conditions: (A) z vector cluster labels and $\mathcal{L}_G(G,D)+\mathcal{L}_{L2}(G)$, (B) random class labels and $\mathcal{L}_G(G,D)+\mathcal{L}_{L2}$, (C) z vector cluster labels and $\mathcal{L}_G(G,D)$, and (D) z vector cluster labels and $\mathcal{L}_{L2}(G)$.

7 Applications

To evaluate the practical usefulness of our proposed method, we use the generated VR stroke datasets (Stage 2 in Fig. 4) in two applications: VR stroke classification and VR stroke prediction. The effectiveness of these applications can also showcase the performance of our model in out-of-distribution generalization.

The development of each application takes a real VR stroke dataset and a desktop dataset as input and involves three steps. First, we train a generative model based on the real VR dataset (i.e., Stage 1 in Fig. 4). Second, we generate new VR strokes by using the trained generator on the desktop dataset (i.e., Stage 2 in Fig. 4). Third, we train VR stroke classifiers and VR stroke predictors on the generated VR dataset with or without the real VR stroke dataset.

7.1 VR Stroke Classification

This application aims to classify Arabic digits (i.e., 0-9) drawn in VR. From the 3DMadLabSD dataset [16], we have excluded 1000 Arabic digit strokes, utilizing the remaining 3000 strokes as the real VR stroke dataset. The DigiLeTs [1] serves as the desktop stroke dataset, comprising 380 instances for each digit. We first train our proposed model (Fig. 4) with the 3000 real VR strokes. Once the generative model is trained, we input instances from the DigiLeTs dataset to obtain synthesized VR digits. Subsequently, we employ both deep learning and template-based classifiers to understand how they respond to synthesized VR digits as their training data.

For deep learning classifiers, we train three PointView-GCN [30] classification models with three datasets: 800 real VR digits from 3DMadLabSD, 800 real VR digits plus 1600 synthesized VR digits, and 1600 synthesized VR digits. Their accuracy on the remaining 200 real VR digits from 3DMadLabSD is 96.67%, 99.63%, and 84.07%, respectively. The results indicate that the synthesized VR strokes can supplement real VR strokes and increase the accuracy of deep learning classifiers. Moreover, the results showcase the potential of our approach to alleviate the burden of collecting VR data, because the model can achieve satisfactory accuracy by relying solely on the synthesized datasets.

For template-based classifiers, we test the five algorithms mentioned in [32], which are designed for VR stroke classification. These algorithms function by matching input data to a set of predefined templates. We integrate the synthesized VR digits with 80 randomly selected real VR digits from 3DMadLabSD, gradually increasing the synthesized data quantity from 0 to 240 instances. As shown in Fig. 9, adding synthesized VR strokes within a certain range improves accuracy across these template-based classifiers. However, a decrease in accuracy is observed when the amount of synthesized data exceeds a threshold, particularly in the case of FreehandUni. This decline may be attributed to the inherent limitations of template-based classifiers. The increased variety brought by the synthesized strokes presents a challenge to their matching capabilities, resulting in less precise classification.

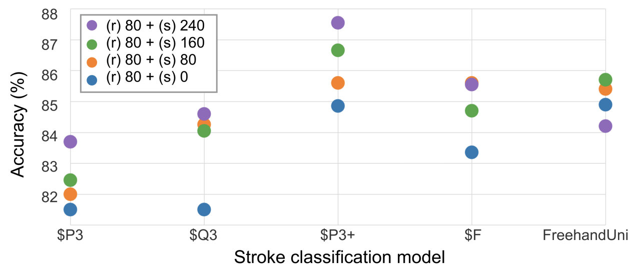


Figure 9: The classification accuracy of five template-based stroke classifiers [32] trained on different combinations of the real VR strokes (r) and synthesized VR strokes (s).

7.2 VR Stroke Prediction

This application aims to predict the next VR stroke a user may want to draw based on strokes s/he has already drawn, which can facilitate VR creation in VR painting software. This application involves sketches with multiple VR strokes but no such datasets are publicly available. Thus, we collect a multi-stroke VR sketch dataset by developing a VR drawing program with logging functions, which can collect controllers' movements with timestamps. With institutional IRB approval, we invite ten users to draw ten sketches (e.g., cats, fish, and envelopes) selected from QuickDraw [2], each ten times. The desktop dataset used in this application is the corresponding ten types of sketches in the QuickDraw dataset. We first train our generative model (Fig. 4) based on our collected VR dataset, consisting of 1000 sketches. Then, we use the trained model and the desktop dataset to generate 5000 instances for each sketch type.

We train a Compositional Stroke Embeddings (CoSE) models [3] to perform stroke prediction. CoSE is an autoencoder model designed for free-form sketches. It projects variable-length strokes into a fixed-dimensional latent space, allowing a relational model to capture the relationship between strokes and predict subsequent strokes. Fig. 10-A and Fig. 11-A show successful examples of a cat and an envelope predicted step by step by the CoSE models trained on 5000 generated instances. Fig. 10-B and Fig. 11-B show failure examples predicted by the CoSE models trained on 100 real sketches collected from our users. The comparison demonstrates that our approach can make the prediction task possible with synthesized VR strokes, although the task is impossible with limited real VR strokes.

8 Discussion

In this paper, we propose and evaluate a convenient, low-cost, and scalable way to obtain diverse and large VR stroke gesture datasets from desktop stroke gestures. This section discusses our reflections, lessons learned, and limitations.

8.1 Reflections on the Use of VR and Desktop Datasets

Our proposed method relies on real VR and desktop stroke datasets to generate VR stroke datasets. We reflect on how to effectively collect and use these datasets in our method to benefit situations when real VR data is scarce and desktop data is out-of-distribution.

Our method does not require collecting real VR and desktop datasets under identical conditions thanks to its generalizability. In our preliminary studies (Sec. 3), we analyze 3DMadLabSD VR stroke dataset [16] and $1 desktop dataset [49]. Our findings suggest that distribution shifts between VR and desktop datasets arise from differences in shapes, speeds, and input devices. Additionally, the two datasets were collected under varied conditions and setups, such as different participant demographics and sensors, which also likely contributed to the observed distribution shifts. In recognition of these known and unknown factors, we do not explicitly divide real VR stroke datasets into several domains based on any single factor (e.g., shapes and users). Instead, we employ advanced mechanisms to identify latent domains and learn domain-invariant representations (Sec. 5). Our method outperforms several baselines that lack these mechanisms (Sec. 6). The proven effectiveness of our method implies its generalizability and robustness even when the setups are different when collecting real VR and desktop datasets. This robustness enhances the applicability of our method, making it more flexible in selecting desktop stroke datasets, which bring diversity regarding commonalities.

A limited real VR dataset that is insufficient for concrete applications might be adequate for training our generative model. One of our digit classifiers, trained solely with generated VR digits, achieves an acceptable accuracy rate of 84.07% (Sec. 7.1). This exemplifies an extreme use case where real VR datasets entirely lack certain patterns. Specifically, the 3000 real VR strokes used to train our generative model do not include digits 0-9. Yet, our method can incorporate desktop digits to synthesize VR digits and enable digit classification. Similarly, predictors trained on small, real VR sketch datasets struggle to perform effectively. However, they deliver satisfactory results when using larger synthesized sketch datasets (Sec. 7.2). In summary, real VR stroke datasets, when limited in diversity or quantity, might be unable to support concrete tasks like VR stroke classification and prediction. Nonetheless, these limited real VR datasets may be adequate for training our generative models. Once trained, the generative models can utilize desktop datasets to synthesize larger and more diverse VR strokes, thereby enabling applications that are infeasible with the limited real VR dataset alone. Though a larger real VR dataset would be ideal for generating better VR strokes (Fig. 7), our method demonstrates the potential to enhance the utility of limited real VR datasets beyond their initial capabilities. It can reduce the burden of collecting extensive VR datasets when resources and time are constrained.

The use of generated VR strokes in downstream applications needs to consider the characteristics of specific algorithms. While deep learning classifiers and predictors benefit from the inclusion of generated VR strokes, we also observe a decline in performance for template-based classifiers when the number of generated strokes exceeds certain thresholds (Fig. 9). This underscores the importance of not assuming a uniform benefit of generated VR strokes across all algorithms. Instead, a crucial step is to first understand the characteristics of specific algorithms, including their suitability for large or small datasets, and their ability to handle diversity and complexity. Moreover, to identify the point at which performance begins to drop, it is necessary to experiment with varying quantities of VR strokes fed into these specific algorithms. These help to ensure that generated VR strokes are effectively and optimally utilized.

8.2 Lessons Learned for Other VR Interaction Data

We distill two lessons for future research to extend our work to generate other types of VR interaction data with desktop data.

Identify appropriate commonalities and additional dimensions for other types of interactions. To generate VR interaction data with desktop data, the first step is to examine whether there are commonalities and additional dimensions between them. Both commonalities and additional dimensions can be explicit or implicit. Explicit commonalities are the direct, measurable actions users undertake, while implicit commonalities capture subtler indicators of user actions, such as intention and emotional response. Explicit additional dimensions involve obvious changes in VR, like head and body movements, while implicit dimensions refer to more nuanced changes, such as an enhanced presence. For instance, if future research wants to generate scan paths for 360-degree images that are not included in existing datasets [36], the commonalities could be users' attention, emotional responses, and points of interest within a fixed viewport. The additional dimensions could be head movements that connect the points of interest across all possible viewports in a 360-degree image.

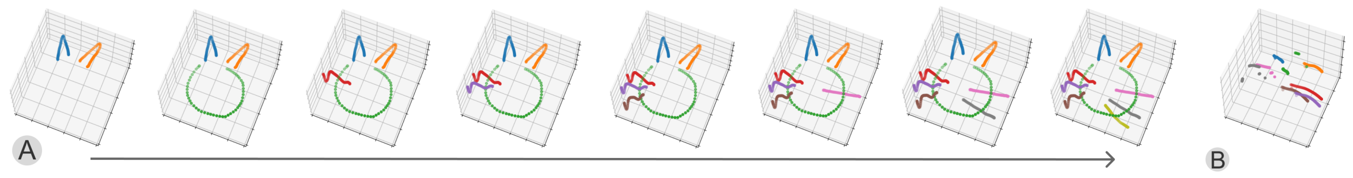


Figure 10: The prediction results of cat sketches. (A) Predicted strokes through stepwise iterations by the CoSE model trained on a large generated VR dataset of cats. (B) Inferior results predicted by the CoSE model trained on the small real VR dataset of cats.

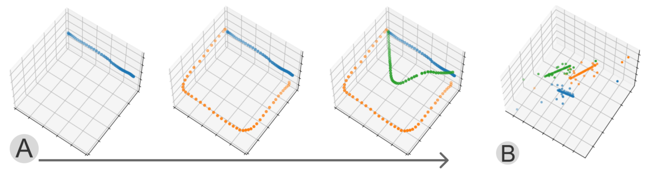


Figure 11: The prediction results of envelope sketches with CoSE models trained on (A) the large generated VR dataset of envelopes, and (B) the small real VR dataset of envelopes.

Pay attention to distribution shifts. Our study reveals that distribution shifts exist between and within VR and desktop stroke datasets. Future research on other types of VR interaction may inspect whether distribution shifts also exist and adopt corresponding solutions to address the challenges caused by the distribution shifts. Specifically, distribution shifts require the models trained on VR interaction datasets to be generalizable to desktop interaction data, which are drawn from unseen distributions. Thus, the solutions may consider incorporating and adapting out-of-distribution generation techniques (Sec. 2.3) based on the data properties (e.g., images, time series, or videos), especially selecting proper ways to characterize the latent distributions and mitigate the distribution gaps.

8.3 Limitations and Future Work

While our approach is effective in generating VR strokes that benefit different applications, we acknowledge some limitations and discuss possible future work to address them.

Evaluation with datasets collected in diverse conditions. The VR stroke datasets used in our evaluation only have positional features (i.e., x, y, z coordinates) and timestamps. However, VR strokes can encompass more complex features. For example, drawing systems like Tilt Brush and GravitySketch use the orientation (i.e., roll, pitch, yaw) of the controllers to determine each stroke's normal and ruling directions. These orientations significantly influence the naturalness and accuracy of VR drawings [34]. To generate VR strokes with these features, a VR stroke dataset capturing controller orientations, stroke normals, and ruling directions is essential. Additionally, a desktop stroke dataset might be beneficial, recording features like the tilt and rotation of a stylus pen, analogous to VR controller orientations. Currently, the absence of publicly available VR and desktop datasets with these features limits our ability to test our generative models in these contexts. Nonetheless, once such VR and desktop datasets become available, our models could be adapted with minimal modifications to generate these features, enhancing the realism and accuracy of VR stroke generation. Similarly, previous research [5] has pointed out that VR strokes collected under different drawing conditions, such as plane settings (e.g., horizontal, vertical, sideways) and visual guidance (e.g., none, stroke, surface, combined), present different depth variations. Currently, we only evaluate our proposed method with datasets of VR strokes drawn on an imaginary vertical plane without visual guidance. However, once VR stroke datasets collected under other conditions are available, our proposed generative model can generate VR strokes that match their distributions with no modifications.

Support for non-planar VR strokes. Our research primarily focuses on planar VR strokes, typically drawn in 3D spaces but representing 2D symbols or objects. While these strokes are common in many applications [35, 47, 50], non-planar VR strokes [6, 8, 28] also play a significant role in fields like industrial design [15, 56]. Unlike planar strokes where z values are regarded as inaccuracies or deviations, non-planar strokes use z values to depict the volume [56] or perspective [14] of an object. Thus, non-planar strokes treat z values as part of target strokes, regardless of VR [27] or desktop [15] stroke datasets. This means that our current network design is not directly applicable to non-planar strokes, because z values are no longer additional dimensions. However, our general idea, namely identifying commonalities and additional dimensions and learning their relationships in out-of-distribution settings, still applies. To extend our model to generate non-planar VR strokes, future work may propose novel reference frames by adopting concepts such as gesture task axes [43] or scaffolds [51, 56], rather than using the traditional Cartesian coordinate system. Then, future work can consider commonalities and additional dimensions under the new reference frames and modify our network design.

9 Conclusion

This paper has demonstrated the feasibility of using desktop strokes as an alternative input source to enrich VR stroke datasets, as well as the usefulness of the proposed method with two applications. Specifically, we formulate the problem of generating VR strokes as a conditional time series generation problem. Then, we propose a two-stage method of first learning the relationships between commonalities and additional dimensions based on VR strokes and then applying the learned relationships to desktop strokes. The generator to learn the relationships consists of discretizing the output space, characterizing latent domains, and learning conditional domain-invariant representations. Taking stroke gestures as a starting point, we believe our methods have much potential and deserve further investigation for other types of interaction.

Acknowledgments

The authors wish to thank the anonymous reviewers for their constructive suggestions. The authors also wish to thank Jipeng Zhang, Tao Yang, Zhiguang Yang, Ziqiang Zheng, and Li Ma for their advice. This work is supported by a General Research Fund (Grant No. 16210321) from the Research Grants Council of Hong Kong.

References

[1] DigiLeTs. `https://github.com/CognitiveModeling/DigiLeTs`, Accessed in 2024.

[2] QuickDraw. `https://github.com/googlecreativelab/quickdraw-dataset`, Accessed in 2024.

[3] E. Aksan, T. Deselaers, A. Tagliasacchi, and O. Hilliges. CoSE: Compositional stroke embeddings. *Advances in Neural Information Processing Systems*, 33:10041–10052, 2020.

[4] L. Anthony, R.-D. Vatavu, and J. O. Wobbrock. Understanding the consistency of users' pen and finger stroke gesture articulation. In *Proceedings of Graphics Interface*, pp. 87–94. 2013.

[5] R. Arora, R. H. Kazi, F. Anderson, T. Grossman, K. Singh, and G. W. Fitzmaurice. Experimental evaluation of sketching on surfaces in VR. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 5643–5654, 2017.

[6] R. Arora and K. Singh. Mid-air drawing of curves on 3D surfaces in virtual reality. *ACM Transactions on Graphics*, 40(3):1–17, 2021.

[7] N. Ashtari, A. Bunt, J. McGrenere, M. Nebeling, and P. K. Chilana. Creating augmented and virtual reality applications: Current practices, challenges, and opportunities. In *Proceedings of the CHI conference on human factors in computing systems*, pp. 1–13, 2020.

[8] C. Chen, M. Yarmand, Z. Xu, V. Singh, Y. Zhang, and N. Weibel. Investigating input modality and task geometry on precision-first 3D drawing in virtual reality. In *IEEE International Symposium on Mixed and Augmented Reality*, pp. 384–393. IEEE, 2022.

[9] B. David-John, K. Butler, and E. Jain. Privacy-preserving datasets of eye-tracking samples with applications in XR. *IEEE Transactions on Visualization and Computer Graphics*, 29(5):2774–2784, 2023.

[10] Y. Du, J. Wang, W. Feng, S. Pan, T. Qin, R. Xu, and C. Wang. AdaRNN: Adaptive learning and forecasting of time series. In *Proceedings of the ACM International Conference on Information & Knowledge Management*, pp. 402–411, 2021.

[11] C. Esteban, S. L. Hyland, and G. Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[12] D. Fei, Z. Gao, L. Yuan, and Z. A. Wen. Collectiar: Computer vision-based word hunt for children with dyslexia. In *Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play*, pp. 171–176, 2022.

[13] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[14] Y. Gryaditskaya, F. Hähnlein, C. Liu, A. Sheffer, and A. Bousseau. Lifting freehand concept sketches into 3D. *ACM Transactions on Graphics*, 39(6):1–16, 2020.

[15] Y. Gryaditskaya, M. Sypesteyn, J. W. Hoftijzer, S. C. Pont, F. Durand, and A. Bousseau. OpenSketch: a richly-annotated dataset of product design sketches. *ACM Transactions on Graphics*, 38(6):232–1, 2019.

[16] J. Huang, P. Jaiswal, and R. Rai. Gesture-based system for next generation natural and intuitive interfaces. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 33(1):54–68, 2019.

[17] S. Hubenschmid, J. Wieland, D. I. Fink, A. Batch, J. Zagermann, N. Elmqvist, and H. Reiterer. ReLive: Bridging in-situ and ex-situ visual analytics for analyzing mixed reality user studies. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2022.

[18] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1125–1134, 2017.

[19] S. M. Lehman, S. Elezovikj, H. Ling, and C. C. Tan. ARCHIE++: A cloud-enabled framework for conducting ar system testing in the wild. *IEEE Transactions on Visualization and Computer Graphics*, 29(4):2102–2116, 2022.

[20] L. A. Leiva, D. Martín-Albo, R. Plamondon, and R.-D. Vatavu. Keytime: Super-accurate prediction of stroke gesture production times. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2018.

[21] L. A. Leiva, D. Martín-Albo, and R.-D. Vatavu. Synthesizing stroke gestures across user populations: A case for users with visual impairments. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 4182–4193, 2017.

[22] X. Li, J.-D. Wang, J. J. Dudley, and P. O. Kristensson. Swarm manipulation in virtual reality. In *Proceedings of the ACM Symposium on Spatial User Interaction*, pp. 1–11, 2023.

[23] Z. Li, Y. Jiang, Y. Zhu, R. Chen, R. Wang, Y. Wang, Y. Yan, and Y. Shi. Modeling the noticeability of user-avatar movement inconsistency for sense of body ownership intervention. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(2):1–26, 2022.

[24] Y. Lin, H. Li, L. Yang, A. Wu, and H. Qu. InkSight: Leveraging sketch interaction for documenting chart findings in computational notebooks. *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[25] Q. Liu, G. Alves, and J. Zhao. Challenges and opportunities for software testing in virtual reality application development. In *Graphics Interface*, 2023.

[26] W. Lu, J. Wang, X. Sun, Y. Chen, and X. Xie. Out-of-distribution representation learning for time series classification. In *The International Conference on Learning Representations*, 2022.

[27] L. Luo, Y. Gryaditskaya, Y. Yang, T. Xiang, and Y.-Z. Song. Fine-grained VR sketching: Dataset and insights. In *International Conference on 3D Vision*, pp. 1003–1013. IEEE, 2021.

[28] M. D. B. Machuca, J. H. Israel, and D. F. Keefe. Toward more comprehensive evaluations of 3D immersive sketching, drawing, and painting. *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[29] D. Martin, A. Serrano, A. W. Bergman, G. Wetzstein, and B. Masia. ScanGAN360: A generative model of realistic scanpaths for 360 images. *IEEE Transactions on Visualization and Computer Graphics*, 28(5):2003–2013, 2022.

[30] S. S. Mohammadi, Y. Wang, and A. Del Bue. Pointview-GCN: 3D shape classification with multi-view point clouds. In *IEEE International Conference on Image Processing*, pp. 3103–3107. IEEE, 2021.

[31] H. Ni, L. Szpruch, M. Wiese, S. Liao, and B. Xiao. Conditional sig-wasserstein gans for time series generation. *arXiv preprint arXiv:2006.05421*, 2020.

[32] M. Ousmer, A. Sluÿters, N. Magrofuoco, P. Roselli, and J. Vanderdonckt. Recognizing 3D trajectories as 2D multi-stroke gestures. *Proceedings of the ACM on Human-Computer Interaction*, 4(ISS):1–21, 2020.

[33] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2536–2544, 2016.

[34] E. Rosales, C. Araújo, J. Rodriguez, N. Vining, D. Yoon, and A. Sheffer. AdaptiBrush: Adaptive general and predictable VR ribbon brush. *ACM Transactions on Graphics*, 40(6):247–1, 2021.

[35] J. Shen, J. J. Dudley, and P. O. Kristensson. Fast and robust mid-air gesture typing for AR headsets using 3D trajectory decoding. *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[36] V. Sitzmann, A. Serrano, A. Pavel, M. Agrawala, D. Gutierrez, B. Masia, and G. Wetzstein. Saliency in VR: How do people explore virtual environments? *IEEE Transactions on Visualization and Computer Graphics*, 24(4):1633–1642, 2018.

[37] Z. Song, J. J. Dudley, and P. O. Kristensson. HotGestures: Complementing command selection and use with delimiter-free gesture-based shortcuts in virtual reality. *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[38] J. Tian, Y. Cao, L. Feng, D. Fu, L. Yuan, H. Qu, Y. Wang, and M. Fan. Poeticar: Reviving traditional poetry of the heritage site of jichang garden via augmented reality. *International Journal of Human-Computer Interaction*, pp. 1–17, 2023.

[39] W. Tong, Z. Chen, M. Xia, L. Y.-H. Lo, L. Yuan, B. Bach, and H. Qu. Exploring interactions with printed data visualizations in augmented reality. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):418–428, 2022.

[40] W. Tong, M. Xia, K. K. Wong, D. A. Bowman, T.-C. Pong, H. Qu, and Y. Yang. Towards an understanding of distributed asymmetric collaborative visualization on problem-solving. In *IEEE Conference Virtual Reality and 3D User Interfaces*, pp. 387–397. IEEE, 2023.

[41] H. Tu, X. Ren, and S. Zhai. A comparative evaluation of finger and pen stroke gestures. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 1287–1296, 2012.

[42] R.-D. Vatavu. Improving gesture recognition accuracy on touch screens for users with low vision. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 4667–4679, 2017.

[43] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock. Relative accuracy measures for stroke gestures. In *Proceedings of the ACM on International Conference on Multimodal Interaction*, pp. 279–286, 2013.

[44] R. Volpi, H. Namkoong, O. Sener, J. C. Duchi, V. Murino, and S. Savarese. Generalizing to unseen domains via adversarial data augmentation. *Advances in Neural Information Processing Systems*, 31, 2018.

[45] J. Wang, C. Lan, C. Liu, Y. Ouyang, T. Qin, W. Lu, Y. Chen, W. Zeng, and P. Yu. Generalizing to unseen domains: A survey on domain generalization. *IEEE Transactions on Knowledge and Data Engineering*,

2022.

[46] W. Wang, S. Liao, F. Zhao, C. Kang, and L. Shao. DomainMix: Learning generalizable person re-identification without human annotations. In *British Machine Vision Conference*, 2021.

[47] Z. Wang, S. Qiu, N. Feng, H. Rushmeier, L. McMillan, and J. Dorsey. Tracing versus freehand for evaluating computer-generated drawings. *ACM Transactions on Graphics*, 40(4):1–12, 2021.

[48] Z. Wang, L. Yuan, L. Wang, B. Jiang, and Z. Wei. Virtuwander: Enhancing multi-modal interaction for virtual tour guidance through large language models. In *Proceedings of the CHI conference on human factors in computing systems*, 2024.

[49] J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 159–168, 2007.

[50] C. Xiao, W. Su, J. Liao, Z. Lian, Y.-Z. Song, and H. Fu. Differsketching: How differently do people sketch 3d objects? *ACM Transactions on Graphics*, 41(6):1–16, 2022.

[51] X. Xu, Y. Zhou, B. Shao, G. Feng, and C. Yu. GestureSurface: VR sketching through assembling scaffold surface with non-dominant hand. *IEEE Transactions on Visualization and Computer Graphics*, 29(5):2499–2507, 2023.

[52] T. Xue, A. El Ali, T. Zhang, G. Ding, and P. Cesar. CEAP-360VR: A continuous physiological and behavioral emotion annotation dataset for 360 VR videos. *IEEE Transactions on Multimedia*, 2021.

[53] T. Yin, L. Hoyet, M. Christie, M.-P. Cani, and J. Pettré. The one-man-crowd: Single user generation of crowd motions using virtual reality. *IEEE Transactions on Visualization and Computer Graphics*, 28(5):2245–2255, 2022.

[54] J. Yoon, D. Jarrett, and M. Van der Schaar. Time-series generative adversarial networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[55] D. Yu, H.-N. Liang, X. Lu, K. Fan, and B. Ens. Modeling endpoint distribution of pointing selection tasks in virtual reality environments. *ACM Transactions on Graphics*, 38(6):1–13, 2019.

[56] X. Yu, S. DiVerdi, A. Sharma, and Y. Gingold. ScaffoldSketch: Accurate industrial design drawing in VR. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 372–384, 2021.

[57] Q. Zhu, L. Yuan, Z. Xu, L. Yang, M. Xia, Z. Wang, H.-N. Liang, and X. Ma. From reader to experiencer: Design and evaluation of a VR data story for promoting the situation awareness of public health threats. *International Journal of Human-Computer Studies*, 181:103137, 2024.