A Optimal Transport and Wasserstein Distances

The Wasserstein-p metric between two probability distributions μ_X and ν_Y is defined as,

$$W_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \mathbf{E}_{(x,y)\sim\gamma} \|x - y\|_p \right)^{1/p}, \tag{12}$$

where $\Gamma(\mu, \nu)$ are all possible joint distributions where $X, Y \in \mathcal{D}$, $(\mathcal{D}, d)$ defines a metric space (here, $d = \|x - y\|_p$) and marginals satisfy $\int_{\mathcal{D}} \gamma(x, y)\mathrm{d}y = \mu(x)$ and $\int_{\mathcal{D}} \gamma(x, y)\mathrm{d}x = \nu(y)$. Note that, while the distance is useful in itself, the optimal transport plan $\gamma*$ is also interesting for some applications.

Given samples from μ and ν, $W_p(\mu, \nu)$ can be computed by solving the optimal transport problem,

$$\gamma^* = \underset{\gamma \in \mathbb{R}_+^{m\times n}}{\operatorname{argmin}} \sum_{i,j} \gamma_{i,j} M_{i,j} \tag{13}$$

$$\text{s.t. } \gamma 1 = [\hat{\mu}]_m; \gamma^T 1 = [\hat{\nu}]_n; \gamma \geq 0 \tag{14}$$

where $[\hat{\mu}]_m$ and $[\hat{\nu}]_n$ represent binned histograms derived from samples from μ and ν with m and n bins respectively. M is a $m \times n$ distance or cost matrix, $M_{i,j}$ represents the cost $d(x, y)$ to transport mass from bin $[\hat{\mu}]_m^i$ to bin $[\hat{\nu}]_n^j$.

It is also possible to avoid binning and compute the Wasserstein distance directly from samples. The problem can then be formulated as,

$$\gamma^* = \underset{\gamma \in \mathbb{R}_+^{n\times n}}{\operatorname{argmin}} \sum_{i,j} \gamma_{i,j} M_{i,j} \tag{15}$$

$$\text{s.t. } \gamma 1 = 1; \gamma^T 1 = 1; \gamma \in \{0, 1\} \tag{16}$$

where $M_{i,j}$ now represents the cost of transporting point x_i to y_j. Each point is considered to be sampled i.i.d. from their respective distributions. Unlike the previous case, we can no longer transport a quanta of a point or mass i.e. fractional assignment is not meaningful, naturally leading to a bipartiteness constraint. This is exactly same as Eq (1) and the optimal transport plan for the problem is a linear sum assignment problem and can be computed using the hungarian algorithm.

B Datasets

We train and evaluate on *Syn2D*, ShapeNet and ModelNet40 datasets.

Syn2D. We generate 2D synthetic point clouds by uniformly sampling 200 points on simple 2D shapes of circles and squares. The circles are generated by uniformly sampling the center and radius from $(0, 1]$. For the squares, we sample its center, rotation and scale uniformly from $[-0.5, 0.5]$, $[0, \frac{\pi}{2}]$, and $[0.5, 1]$, respectively. We refer to the synthetic dataset as *Syn2D* (Fig. 2).

ShapeNet. ShapeNet Chang et al. (2015) is a richly-annotated, large-scale dataset of 3D shapes. We train and evaluate our models using point clouds from one of the three categories in the ShapeNet dataset : airplane, chair, and car. We sample 1024 points from the ShapeNet point clouds for training and evaluate on a range of point cloud sizes.

ModelNet40. Chang et al. (2015) introduced the ModelNet project to provide a comprehensive and clean collection of $3D$ CAD models for objects, compiled using a list of the most common object categories. In our experiments, we evaluate our models trained with a single ShapeNet category on the ModelNet40 dataset. ModelNet40 has 40 different categories.

ScanObjectNN. Uy et al. (2019) proposed the ScanObjectNN dataset, a real-world point cloud object dataset based on scanned indoor scene data, in order to provide a more realistic benchmark as compared to ModelNet40. In our experiments, we evaluate our models trained with a single ShapeNet category on pairs of point clouds derived from the training set of ScanObjectNN dataset.

Table 3 provides a summary and statistics about of the datasets.

Table 3: Summary of datasets. The top five rows show statistics about the original point cloud datasets, while the bottom four rows show statistics about the pair point cloud dataset that was used for our evaluation and/or training.

	SYN2D	MODELNET40	SCANOBJECTNN	SHAPENET	AIRPLANE	CAR	CHAIR
# CATEGORIES	2	40	15	55	1	1	1
FEATURE DIM	2	3	3	3	3	3	3
CARDINALITY	200	2048	2048	15000	15000	15000	15000
# TRAIN SAMPLES	8000	9840	2309	35708	2832	2458	4612
# VAL SAMPLES	2000	2468	581	5158	405	352	662
# TRAIN PAIR SAMPLES	20000	-	-	-	10000	10000	10000
TRAIN CARDINALITY	200	-	-	-	1024	1024	1024
# VAL PAIR SAMPLES	5000	2000	10000	-	2000	2000	2000
VAL CARDINALITY	200	1024	1024	-	256-8192	256-8192	256-8192

Further, we build pairs (U, V) of point clouds by randomly sampling pairs from the train or validation splits of point clouds datasets summarized in Table . We refer the first argument in the pair U as the *source* and the second argument V as *target*.

B.1 AUGMENTATIONS

In order to improve generalization, we augment the datasets with point cloud pairs. We randomly sample a point cloud pair (U, V) from the dataset and another noisy point cloud N by randomly sampling points from $\mathcal{N}(0, 1)$ and scaling the whole point cloud by $\sigma \sim \mathcal{U}(0.1, 1.1)$. Further, we augment the point cloud pairs according to the following schemes:

- (U, V) : the originally sampled pair.
- (U, N) : target is replaced by the noisy point cloud.
- $(U, U + N)$: target is a corrupted version of U with additive noise N.
- $(U, \tilde{U} + N)$: $\tilde{U}$ denotes a point cloud which is similar to U. For *Syn2D*, we perturb the surface parameters (radius, scale, center, etc.) used for sampling U and sample points on the perturbed surface. For ShapeNet, we independently sample different set of points from the original surface.
- $(U, V + N)$: target is a corrupted version of a randomly sampled point cloud from the dataset.

The resulting dataset constitutes 20% samples from each of the above splits. Validation splits are generated randomly and independently for *Syn2D*, while for ShapeNet and ModelNet40, we use the validation split provided. We also augment the validation split with the same scheme as discussed above. The specifics of the pair point-cloud dataset is summarized in Table 3.

C MODELS

Transformers. Let $X = [x_1, \cdots, x_m] \in \mathbb{R}^{m \times d_{\text{model}}}$ be the input sequence (or set) of m vectors. A transformer layer performs the following computation Vaswani et al. (2017) :

$$X' = \text{LN}\left(X + \text{Multihead}(X)\right)$$
$$t_l(X) = \text{LN}\left(X' + \text{FFN}(X')\right)$$

where LN and FFN stand for layer norm and feed-forward network, respectively. Multihead denotes a multi-head attention layer which allows the model to jointly attend to information from different representation subspaces at different positions. It consists of a stack of H scaled dot-product attention layers and computes key, query and value matrices, followed by a softmax as follows :

$$K_h = XW_h^K, \quad Q_h = XW_h^Q, \quad V_h = XW_h^V$$
$$A_h = \text{soft-max}\left(\frac{Q_h {K_h}^\top}{d_k}\right) V_h$$
$$\text{MultiHead}(X) = \bar{A} = \text{concat}\left(A_1, \cdots, A_H\right) W_O$$

The final output of the encoder can be written as a composition:

$$t(X) = t_N\left(t_{N-1}\left(\ldots\left(t_1(X)\right)\right)\right)$$

MLP. The baseline MLP model predicts directly the distance and it does not use the matching information. It has about $110K$ parameters in total. The point-wise MLP backbone $g(.)$ is composed of three hidden layer of sizes $4, 8$ and 16, with ReLU non-linearity. It outputs a single embedding of dimension 128 for each point cloud after aggregating the point level features. The embedding of the point clouds are then concatenated and passed to the prediction head which is also an MLP with four hidden layers of sizes $256, 128, 64$ and 16, and outputs a single scalar which is interpreted as the predicted distance.

DeepEMD. We use a transformer encoder backbone which transforms the raw input point clouds into contextualized point level features. The transformer encoder is followed by the output layer which computes the queries and keys and finally the attention matrix which is interpreted as the matching as explained in § 3.2. The model constitutes about $803K$ learnable paramters, with 8 transformer encoder layers, each with 6 heads. The latent dimensions ($d_{\text{model}}, d_{\text{keys}}$, etc.) for each layer were all set to 78.

We use the ADAM optimizer with a constant learning rate of 0.001 for DeepEMD and 0.0001 for the MLP.

D SINKHORN DISTANCE

The Sinkhorn distance Cuturi (2013) considers a regularized OT optimization problem :

$$\gamma^* = \underset{\gamma \in \mathbb{R}_+^{m \times n}}{\operatorname{argmin}} \sum_{i,j} \gamma_{i,j} M_{i,j} + \lambda \Omega(\gamma)$$
$$\text{s.t. } \gamma 1 = [\hat{\mu}]_m; \gamma^T 1 = [\hat{\nu}]_n; \gamma \geq 0$$

where λ is the regularization coefficient and $O(\gamma) = \sum_{i,j} \gamma_{i,j} \log(\gamma_{i,j})$ is a entropy regularization term which makes the optimization problem smooth and strictly convex allowing for optimization procedures such as the Sinkhorn-Knopp algorithm. In this paper, we use the Python Optimal Transport (POT) python library Flamary et al. (2021) for computing the Sinkhorn distances. It is an iterative algorithm and can be evaluated in $O(N^2)$ time complexity.

In our experiments, we used a regularization multiplier of 0.1. Reducing the multiplier typically resulted in numerical issues for a larger portion of the samples, whereas increasing it led to greater inaccuracies in estimation with the same computation time (for a fixed number of iterations). We show the performance metrics with different regularisation parameters in Table 4.

Table 4: Effect of regularization parameter on Sinkhorn's algorithm (100 iterations). Results on ScanObjectNN dataset with 10000 samples. The second row shows number of instances which had numerical issues.

λ	0.08	0.1	0.12	0.14
# FAIL	686	80	6	0
r	0.616	0.929	0.993	0.999
ρ	0.708	0.965	0.997	0.999
τ	0.785	0.968	0.988	0.986
$\text{RE}_{0.1}$	0.033	0.038	0.045	0.052
$\text{RE}_{0.5}$	0.068	0.078	0.093	0.110
$\text{RE}_{0.9}$	0.288	0.244	0.166	0.342
$\text{CS}_{0.1}$	0.815	0.879	0.869	0.851
$\text{CS}_{0.5}$	0.991	0.992	0.991	0.989
$\text{CS}_{0.9}$	0.999	1.0	0.999	0.999
ACCURACY	21.29	20.049	18.49	17.22
B	20.72	19.439	18.09	17.03
B_{corr}	10.97	9.961	8.98	8.23

The difficulty of choosing the regularisation parameter also makes it difficult for training generative models with Sinkhorn as it often leads to numerical issues. We experimented with several variants

of Sinkhorn, including log-space Sinkhorn, log-stabilized Sinkhorn, Greenkhorn, etc. However, we discovered that they either exhibited significantly slower performance compared to the vanilla variant, frequently failed to converge to a satisfactory optimal transport matrix within a finite timeframe, or sometimes exhibited both issues.

E Extended Results

E.1 Distance and Matching Estimation

We compare performance of different models and metrics in Table 5. The models were trained on a single ShapeNet category and evaluated on the validation split of the same category. The numbers are averaged over all training categories as well as four training seeds.

Table 5: Performance comparison of different metrics and models. The models are trained on a single ShapeNet category and evaluated on the test split of the same category. The reported numbers are averaged over all categories and four training seeds. The first six rows show distance estimation metrics (see § 4.2), while the last six rows correspond to matching estimation metrics. The arrows next to the metrics indicate whether higher (↑) or lower (↓) values are better. Chamfer and Sinkhorn are deterministic, thus variances are not reported. Further, our MLP model does not provide accuracy and bipartiteness metrics.

MODEL	CHAMFER	SINKHORN	MLP (OURS)	DEEPEMD (OURS)
r (↑)	0.963	0.995	0.998 ± 0.0	$\mathbf{1.0} \pm 0.0$
ρ (↑)	0.953	0.997	0.998 ± 0.001	$\mathbf{1.0} \pm 0.0$
τ (↑)	0.827	0.987	0.966 ± 0.003	$\mathbf{0.988} \pm 0.001$
$\text{RE}_{0.1}$ (↓)	0.023	0.051	$\mathbf{0.002} \pm 0.0$	0.007 ± 0.003
$\text{RE}_{0.5}$ (↓)	0.109	0.106	$\mathbf{0.015} \pm 0.001$	0.017 ± 0.005
$\text{RE}_{0.9}$ (↓)	0.31	0.271	0.076 ± 0.006	$\mathbf{0.032} \pm 0.005$
$\text{CS}_{0.1}$ (↑)	-0.173	0.831	-0.034 ± 0.049	$\mathbf{0.964} \pm 0.001$
$\text{CS}_{0.5}$ (↑)	0.85	0.986	0.798 ± 0.018	$\mathbf{1.0} \pm 0.0$
$\text{CS}_{0.9}$ (↑)	0.998	0.999	0.974 ± 0.003	$\mathbf{1.0} \pm 0.0$
ACCURACY (↑)	11.677	28.407	-	$\mathbf{64.648} \pm 0.404$
B (↑)	17.784	31.889	-	$\mathbf{75.896} \pm 0.521$
B_{corr} (↑)	5.626	16.658	-	$\mathbf{55.719} \pm 0.568$

Tables 6 and 7 show the per-category performance comparison, averaged over four training seeds.

Table 6: Per-category distance estimation performance measures of different models and metrics when train and test category are same. The reported number are averaged over four training seeds.

		r	ρ	τ	$\text{RE}_{0.1}$	$\text{RE}_{0.5}$	$\text{RE}_{0.9}$
TRAIN_CATE	MODEL/METRIC						
AIRPLANE	CHAMFER	0.9797	0.9647	0.8519	0.0141	0.0962	0.3018
	DEEPEMD	$\mathbf{0.9998} \pm 0.0$	$\mathbf{0.9997} \pm 0.0$	0.9879 ± 0.001	0.0039 ± 0.0014	0.0142 ± 0.0031	$\mathbf{0.0306} \pm 0.004$
	MLP	0.9992 ± 0.0001	0.9986 ± 0.0001	0.9722 ± 0.0018	$\mathbf{0.002} \pm 0.0003$	$\mathbf{0.0125} \pm 0.0018$	0.0772 ± 0.0038
	SINKHORN	0.9998	0.9997	**0.9881**	0.0496	0.1104	0.2984
CAR	CHAMFER	0.9675	0.9564	0.8318	0.035	0.1167	0.324
	DEEPEMD	$\mathbf{0.9997} \pm 0.0001$	$\mathbf{0.9998} \pm 0.0$	$\mathbf{0.9891} \pm 0.0006$	0.006 ± 0.0047	0.0156 ± 0.0068	$\mathbf{0.0302} \pm 0.0097$
	MLP	0.9993 ± 0.0002	0.9988 ± 0.0004	0.9746 ± 0.0035	$\mathbf{0.0018} \pm 0.0002$	$\mathbf{0.0112} \pm 0.001$	0.0625 ± 0.012
	SINKHORN	0.991	0.9941	0.9854	0.0531	0.1124	0.2986
CHAIR	CHAMFER	0.9431	0.9382	0.7976	0.0192	0.1133	0.3037
	DEEPEMD	$\mathbf{0.9997} \pm 0.0$	$\mathbf{0.9997} \pm 0.0001$	0.9866 ± 0.0013	0.0103 ± 0.0088	0.0225 ± 0.0118	$\mathbf{0.0356} \pm 0.012$
	MLP	0.9965 ± 0.0013	0.9959 ± 0.0015	0.9504 ± 0.0079	$\mathbf{0.0037} \pm 0.0006$	$\mathbf{0.0214} \pm 0.0023$	0.0881 ± 0.0124
	SINKHORN	0.9942	0.9969	**0.9886**	0.049	0.0955	0.2167

E.2 Out-of-Distribution Generalization

Table 8 shows the out-of-distribution generalization performance for our models. The trained model on a particular ShapeNet category is evaluated on the validation split of other ShapeNet categories.

Table 7: Per-category matching estimation performance measures of different models and metrics when train and test category are same. The reported number are averaged over four training seeds.

		$CS_{0.1}$	$CS_{0.5}$	$CS_{0.9}$	ACCURACY	B	B_CORR
TRAIN_CATE	MODEL/METRIC						
AIRPLANE	CHAMFER	-0.0813	0.8446	0.9973	10.1768	16.7437	4.7461
	DEEPEMD	$\mathbf{0.9643} \pm 0.0027$	$\mathbf{1.0} \pm \mathbf{0.0}$	$\mathbf{1.0} \pm 0.0$	$\mathbf{61.9119} \pm 0.9043$	$\mathbf{73.3128} \pm 1.1746$	$\mathbf{52.2082} \pm 1.246$
	MLP	-0.0492 ± 0.0533	0.7766 ± 0.0263	0.9722 ± 0.003	-	-	-
	SINKHORN	0.8314	0.9871	0.9994	25.2956	29.018	13.9732
CAR	CHAMFER	-0.246	0.8615	1.0	13.7079	20.7186	6.8617
	DEEPEMD	$\mathbf{0.9585} \pm 0.0025$	$\mathbf{1.0} \pm \mathbf{0.0}$	$\mathbf{1.0} \pm 0.0$	$\mathbf{67.7243} \pm 0.6554$	$\mathbf{78.7286} \pm 0.797$	$\mathbf{59.6723} \pm 0.9245$
	MLP	0.017 ± 0.0652	0.8388 ± 0.0187	0.9804 ± 0.0029	-	-	-
	SINKHORN	0.8043	0.9845	0.9993	31.1621	35.5971	19.3105
CHAIR	CHAMFER	-0.1929	0.8442	0.9976	11.145	15.8883	5.2694
	DEEPEMD	$\mathbf{0.9703} \pm 0.0007$	$\mathbf{1.0} \pm \mathbf{0.0}$	$\mathbf{1.0} \pm 0.0$	$\mathbf{64.3079} \pm 0.4712$	$\mathbf{75.6459} \pm 0.657$	$\mathbf{55.2757} \pm 0.7011$
	MLP	-0.0695 ± 0.1211	0.7793 ± 0.0438	0.97 ± 0.0072	-	-	-
	SINKHORN	0.8558	0.9876	0.9992	28.7644	31.0521	16.69

The numbers are averaged over these other categories as well as four training seeds. Tables 9 and 10 show the performance in the same setting but for each test category separately.

Table 8: Out-of-distribution (category) generalization for our models and comparison with other metrics (Chamfer and Sinkhorn). The models are trained on a single ShapeNet category and evaluated on other ShapeNet categories. The reported numbers are averaged over these categories as well as four training seeds. The first five rows show distance estimation metrics (see § 4.2), while the last five rows correspond to matching estimation metrics. The arrows next to the metrics indicate whether higher (↑) or lower (↓) values are better.

MODEL	MLP (OOD)	MLP	DEEPEMD (OOD)	DEEPEMD
r (↑)	0.98 ± 0.019	0.998 ± 0.001	0.999 ± 0.001	1.0 ± 0.0
ρ (↑)	0.976 ± 0.02	0.998 ± 0.001	0.999 ± 0.0	1.0 ± 0.0
τ (↑)	0.886 ± 0.04	0.966 ± 0.004	0.977 ± 0.003	0.988 ± 0.001
$RE_{0.1}$ (↓)	0.014 ± 0.003	0.002 ± 0.0	0.009 ± 0.009	0.007 ± 0.005
$RE_{0.5}$ (↓)	0.065 ± 0.018	0.015 ± 0.002	0.024 ± 0.012	0.017 ± 0.007
$RE_{0.9}$ (↓)	0.319 ± 0.132	0.076 ± 0.009	0.05 ± 0.011	0.032 ± 0.008
$CS_{0.1}$ (↑)	-0.208 ± 0.089	-0.034 ± 0.074	0.933 ± 0.004	0.964 ± 0.002
$CS_{0.5}$ (↑)	0.714 ± 0.047	0.798 ± 0.027	1.0 ± 0.0	1.0 ± 0.0
$CS_{0.9}$ (↑)	0.963 ± 0.006	0.974 ± 0.004	1.0 ± 0.0	1.0 ± 0.0
ACCURACY (↑)	-	-	54.35 ± 1.16	64.648 ± 0.606
B (↑)	-	-	67.922 ± 1.343	75.896 ± 0.782
B_{corr} (↑)	-	-	44.293 ± 1.445	55.719 ± 0.851

Table 9: Per-category distance estimation performance measures of different models and metrics when train and test category are different. The reported number are averaged over four training seeds.

		r	ρ	τ	$RE_{0.1}$	$RE_{0.5}$	$RE_{0.9}$
TRAIN_CATE	TEST_CATE						
AIRPLANE	AIRPLANE	0.9998 ± 0.0	0.9997 ± 0.0	0.9879 ± 0.001	0.0039 ± 0.0014	0.0142 ± 0.0031	0.0306 ± 0.004
	CAR	0.9989 ± 0.0006	0.9997 ± 0.0001	0.9864 ± 0.001	0.0035 ± 0.0005	0.016 ± 0.0036	0.0308 ± 0.0039
	CHAIR	0.9981 ± 0.0003	0.9983 ± 0.0003	0.9689 ± 0.0016	0.0044 ± 0.0011	0.0212 ± 0.0033	0.0491 ± 0.0048
CAR	AIRPLANE	0.9993 ± 0.0002	0.9989 ± 0.0003	0.9766 ± 0.0017	0.0092 ± 0.0065	0.0256 ± 0.011	0.0634 ± 0.0114
	CAR	0.9997 ± 0.0001	0.9998 ± 0.0	0.9891 ± 0.0006	0.006 ± 0.0047	0.0156 ± 0.0068	0.0302 ± 0.0097
	CHAIR	0.9978 ± 0.0001	0.9981 ± 0.0002	0.9671 ± 0.0017	0.0045 ± 0.0019	0.0206 ± 0.006	0.0525 ± 0.0022
CHAIR	AIRPLANE	0.999 ± 0.0004	0.9983 ± 0.0008	0.9751 ± 0.0047	0.0164 ± 0.0114	0.034 ± 0.0127	0.0631 ± 0.0092
	CAR	0.9987 ± 0.0007	0.9996 ± 0.0001	0.9867 ± 0.0007	0.0141 ± 0.0116	0.0275 ± 0.014	0.0422 ± 0.0139
	CHAIR	0.9997 ± 0.0	0.9997 ± 0.0001	0.9866 ± 0.0013	0.0103 ± 0.0088	0.0225 ± 0.0118	0.0356 ± 0.012

E.3 DEEPEMD AS A LOSS

Fig. 9 shows more samples with the input point cloud and the reconstructed output from SetVAE when trained with EMD, Chamfer or DeepEMD as the reconstruction loss.

Table 10: Per-category matching estimation performance measures of different models and metrics when train and test category are different. The reported number are averaged over four training seeds.

		$CS_{0.1}$	$CS_{0.5}$	$CS_{0.9}$	ACCURACY	B	B_CORR
TRAIN_CATE	TEST_CATE						
AIRPLANE	AIRPLANE	0.9643 ± 0.0027	1.0 ± 0.0	1.0 ± 0.0	61.9119 ± 0.9043	73.3128 ± 1.1746	52.2082 ± 1.246
	CAR	0.9347 ± 0.004	1.0 ± 0.0	1.0 ± 0.0	61.4086 ± 0.9383	72.6142 ± 1.3357	51.6545 ± 1.3383
	CHAIR	0.926 ± 0.0045	0.9994 ± 0.0001	1.0 ± 0.0	48.9237 ± 0.9275	63.1441 ± 1.0583	38.1275 ± 1.1302
CAR	AIRPLANE	0.9279 ± 0.0027	0.9997 ± 0.0	1.0 ± 0.0	50.7472 ± 1.1176	66.2 ± 1.0652	40.6049 ± 1.3128
	CAR	0.9585 ± 0.0025	1.0 ± 0.0	1.0 ± 0.0	67.7243 ± 0.6554	78.7286 ± 0.797	59.6723 ± 0.9245
	CHAIR	0.9218 ± 0.0035	0.9994 ± 0.0001	1.0 ± 0.0	47.7712 ± 1.1373	63.8324 ± 1.1691	37.6786 ± 1.2958
CHAIR	AIRPLANE	0.9401 ± 0.0031	0.9998 ± 0.0	1.0 ± 0.0	53.7969 ± 0.8777	67.1421 ± 1.0913	43.3759 ± 1.0926
	CAR	0.945 ± 0.0006	1.0 ± 0.0	1.0 ± 0.0	63.4508 ± 0.5761	74.5965 ± 0.7858	54.3188 ± 0.8313
	CHAIR	0.9703 ± 0.0007	1.0 ± 0.0	1.0 ± 0.0	64.3079 ± 0.4712	75.6459 ± 0.657	55.2757 ± 0.7011

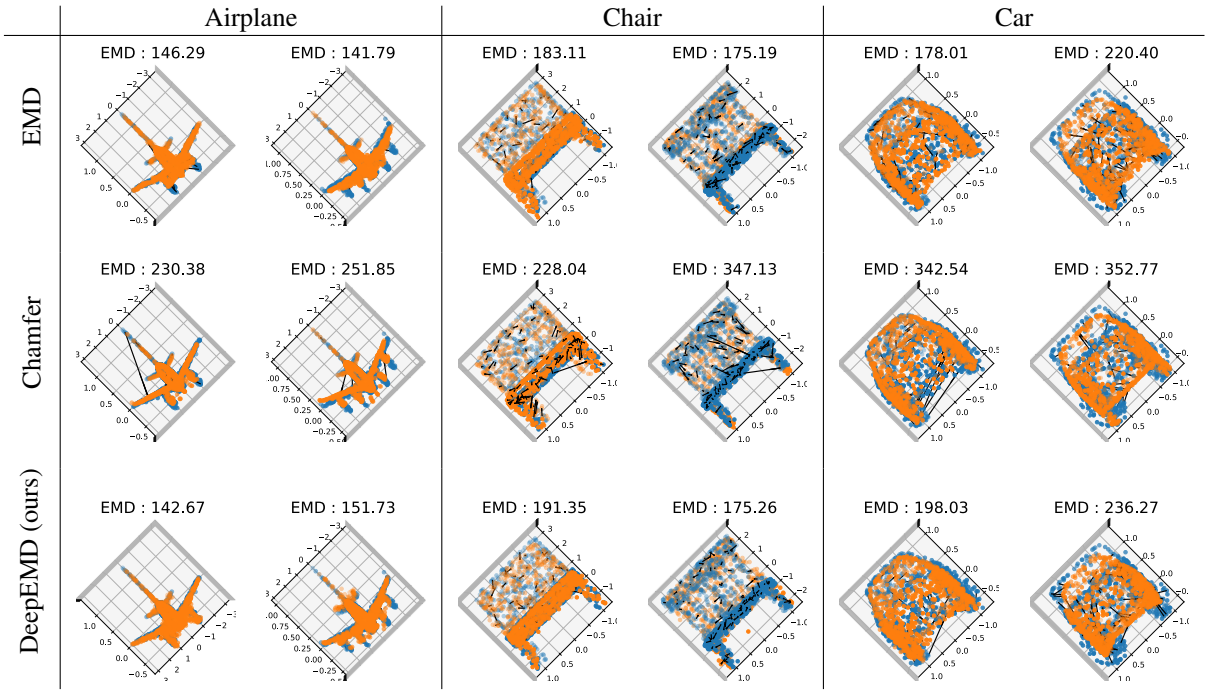

Figure 9: Reconstruction on validation data with SetVAE trained with differnt reconstruction losses : EMD (top), Chamfer (middle) and DeepEMD surrogate (bottom). Training with DeepEMD as a loss consistently achieves lower reconstruction EMD as compared to Chamfer loss.