昆明理工大學
KUNMING UNIVERSITY OF SCIENCE AND TECHNOLOGY

OptDE论文汇报

林涛
昆明理工大学
2025年7月2日

汇报提纲

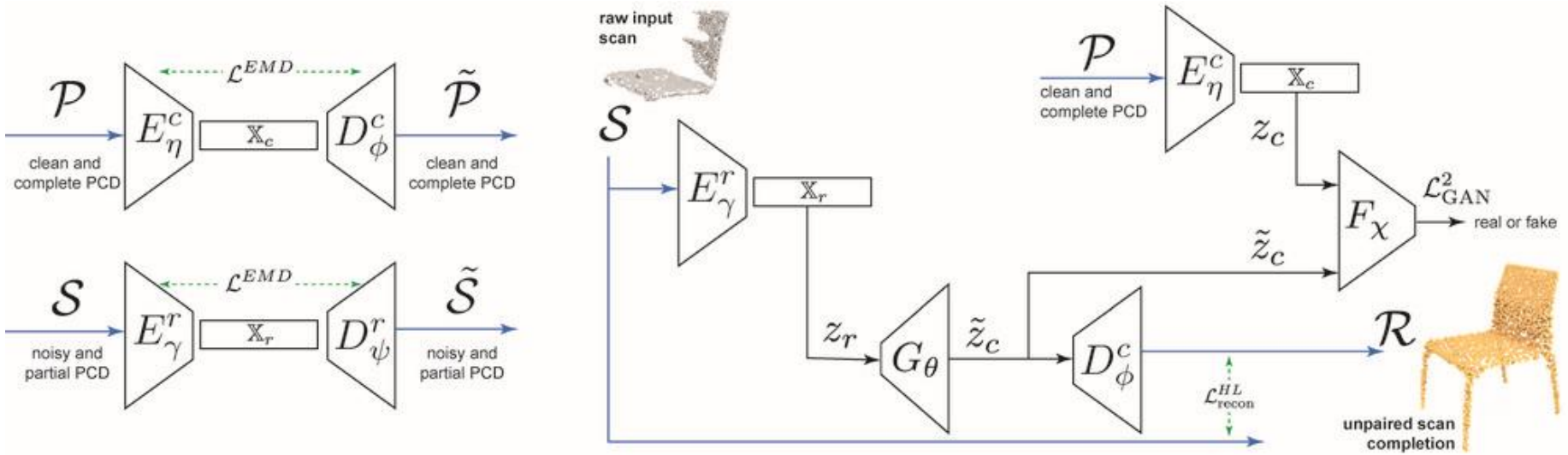
论文解读

代码复现

改进与分析

无监督点云补全

[ICLR 2020] Unpaired point cloud completion on real scans using adversarial training



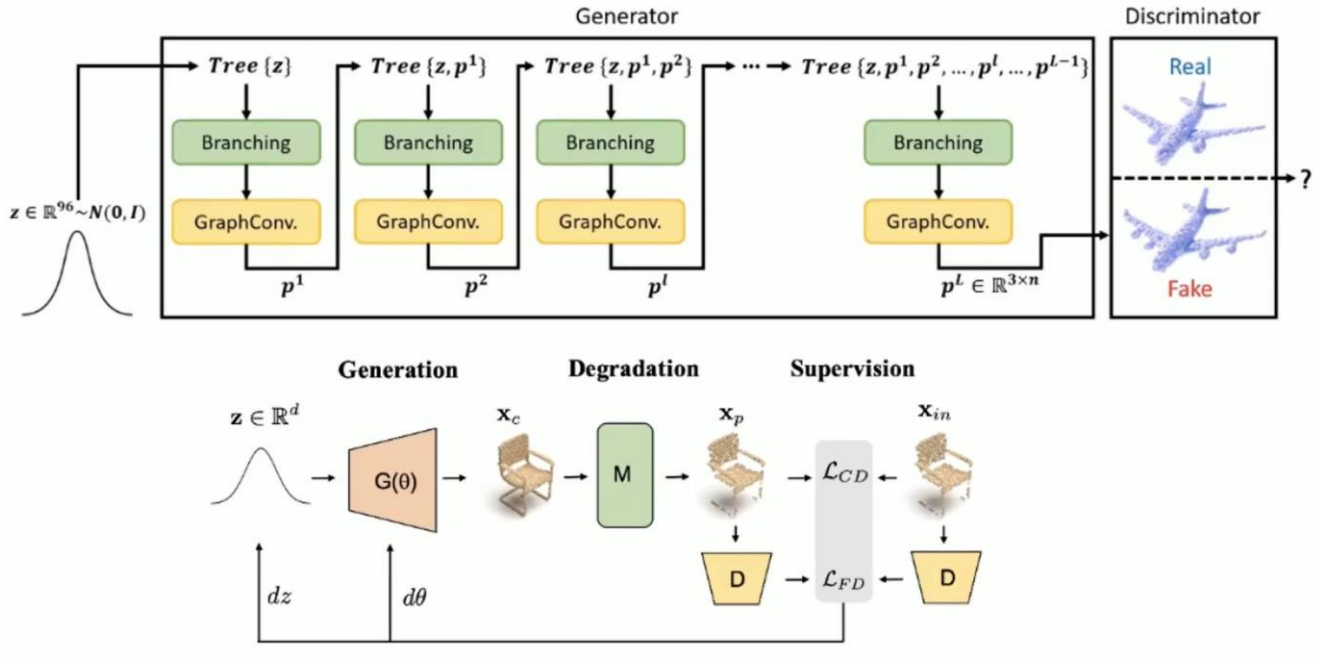
关键思想：

- 基于对抗训练的非配对点云补全方法，无需配对的完整与残缺点云数据。
- **忽略问题：补全任务中预测的完整点云应该与输入部分形状的域相对应。**

Chen, X., Chen, B., Mitra, N.J.: Unpaired point cloud completion on real scans using adversarial training. **ICLR** 2020

无监督点云补全

[CVPR 2021] Unsupervised 3D Shape Completion through GAN Inversion



关键思想：

- 基于 TreeGAN 和生成对抗网络（GAN）反演的ShapeInversion方法，在优化阶段对在虚拟形状上训练的解码器进行微调，从而提升了性能。
- **当输出域差异非常大时，仅依靠优化阶段的适配是不够的。**

Zhang, Junzhe, et al. Unsupervised 3d shape completion through gan inversion. CVPR. 2021.

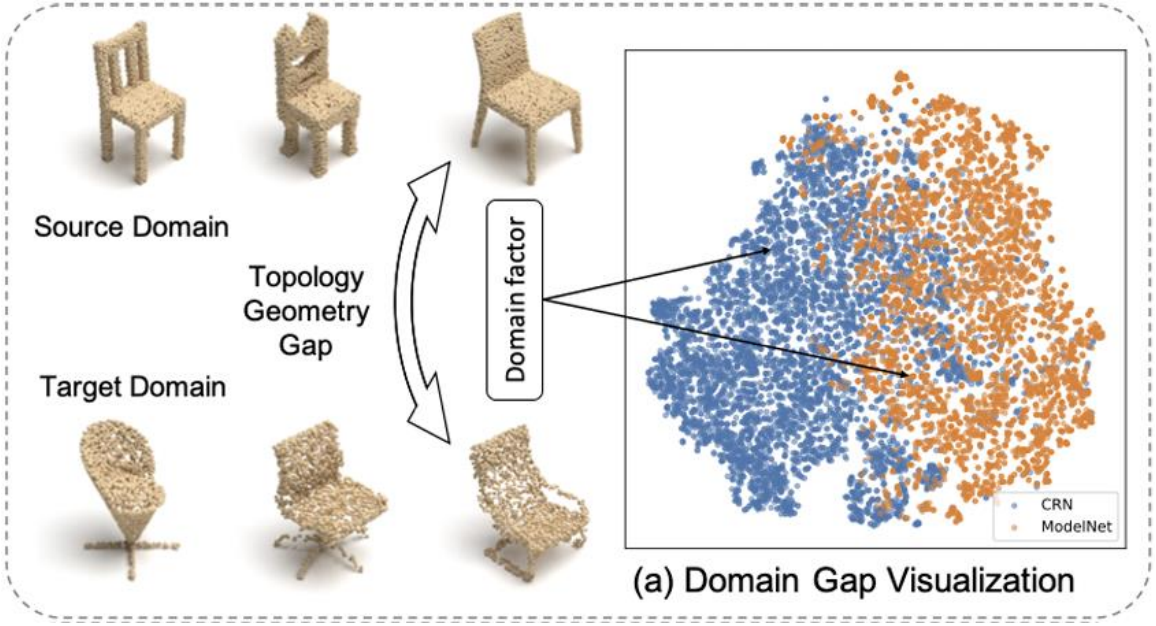
真实场景中的核心挑战

□ 核心挑战：**域差距**，即虚拟训练数据与真实扫描数据在几何、拓扑和特征分布上的显著差异

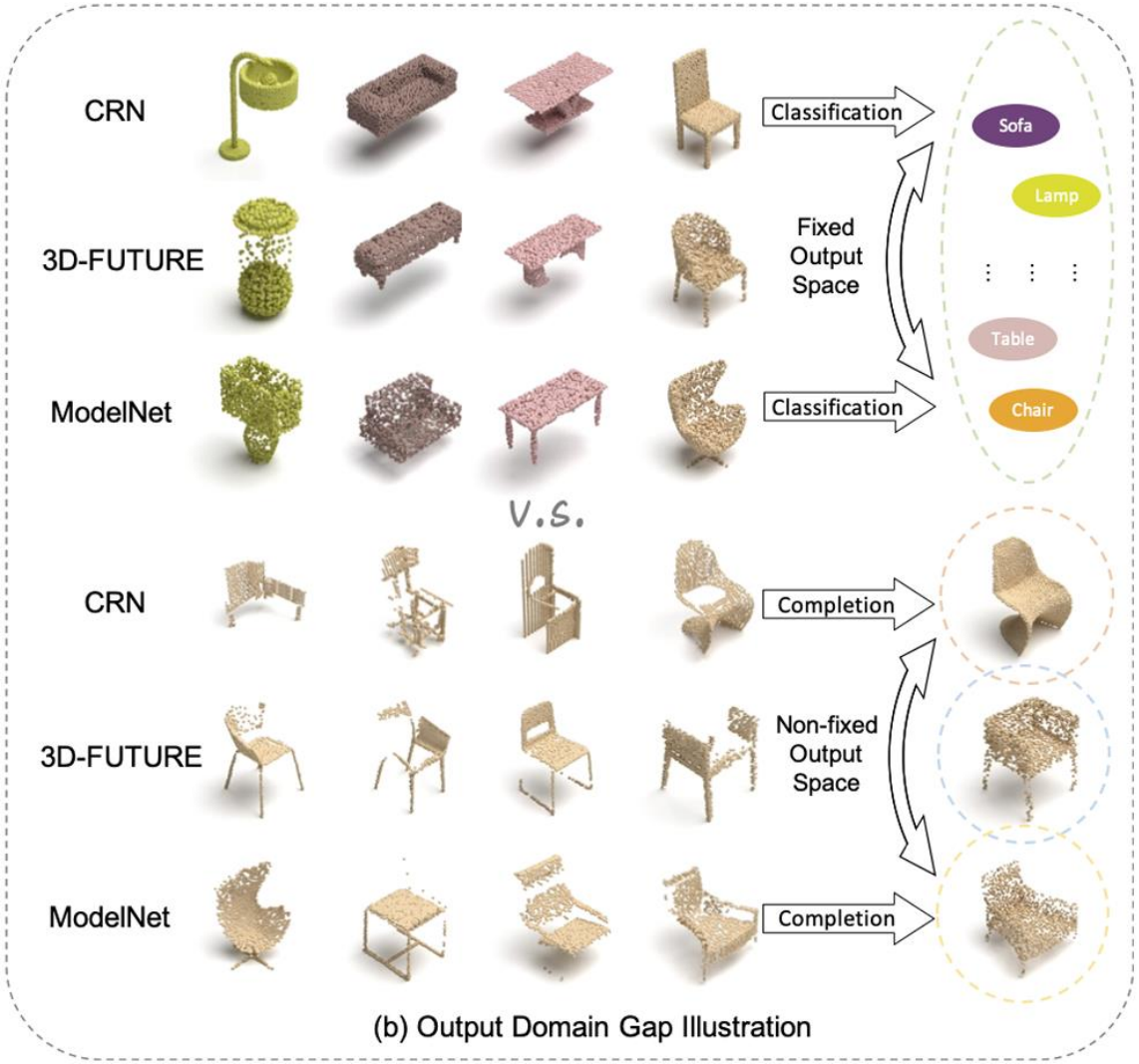
□ 真实数据问题：

1. **输入域差距**：真实点云含扫描噪声、局部缺失不规则、密度不均，而虚拟数据规则无噪声

2. **输出域差距**：补全任务要求预测的完整点云与输入同属真实域，但传统解码器在虚拟数据上训练，生成的形状过度光滑、对称，与真实物体的粗糙表面、非对称结构不匹配



输出域差距作为补全任务的特有先验



(b) Output Domain Gap Illustration

- **分类任务：跨域分类任务的输出空间（如类别标签）是域不变的，只需保证特征提取器对域不敏感**

- **补全任务：补全任务中，若输入是真实扫描的部分点云，理想的完整点云应具有真实域的拓扑细节（如磨损表面），但传统解码器生成的虚拟域完整形状（如光滑表面）与真实域特征不匹配，形成输出域差距**

点云解纠缠为因子

□ 设计动机：同一类别的不同域完整形状共享语义部件，但**拓扑和几何模式不同**，因此分离域因子与形状因子；部分点云的缺失由扫描视角导致，因此**引入遮挡因子表示视角信息**。

□ 思路：

- 补全时将遮挡因子置零，保留域因子和形状因子，使生成的完整点云既包含目标域特征（如真实粗糙表面），又保留正确语义形状
- 因子置换一致性正则化确保三者独立，避免信息耦合

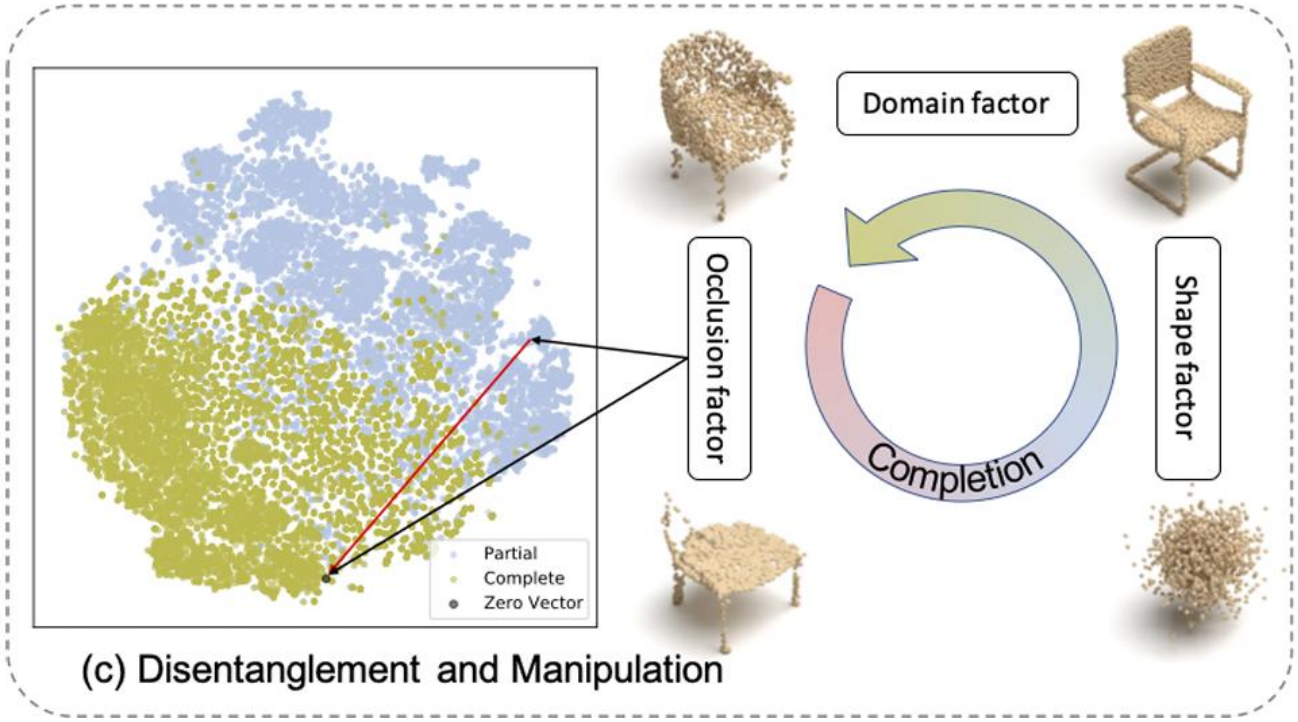
点云解纠缠为因子

□ 三个因子

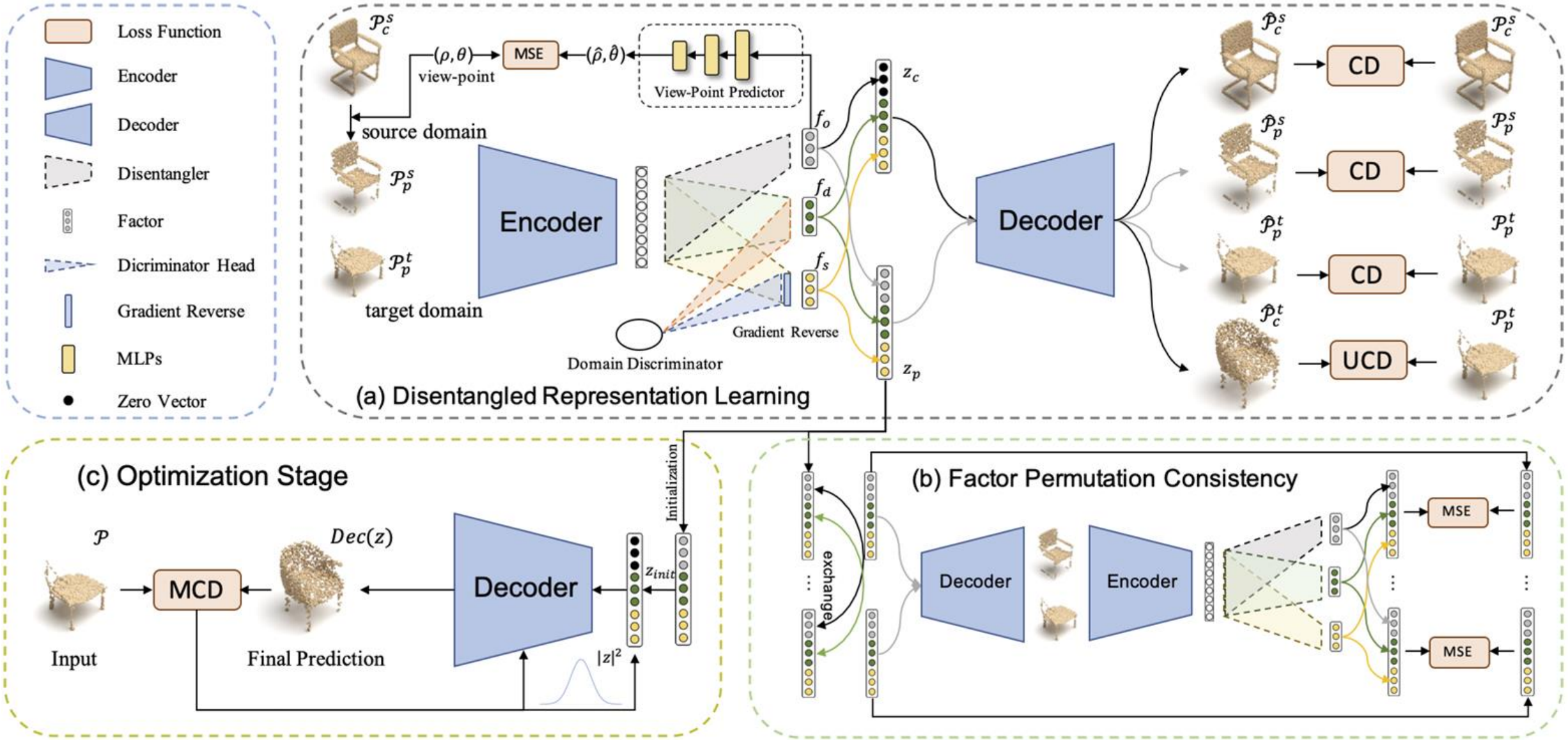
遮挡因子f_o: 表征扫描视角引起的局部缺失，由视点预测任务自监督学习；

域因子f_d: 表征数据所属域（虚拟 / 真实）的特征，通过域判别器提取；

形状因子f_s: 域无关的语义形状特征，通过对抗学习确保其不包含域信息。



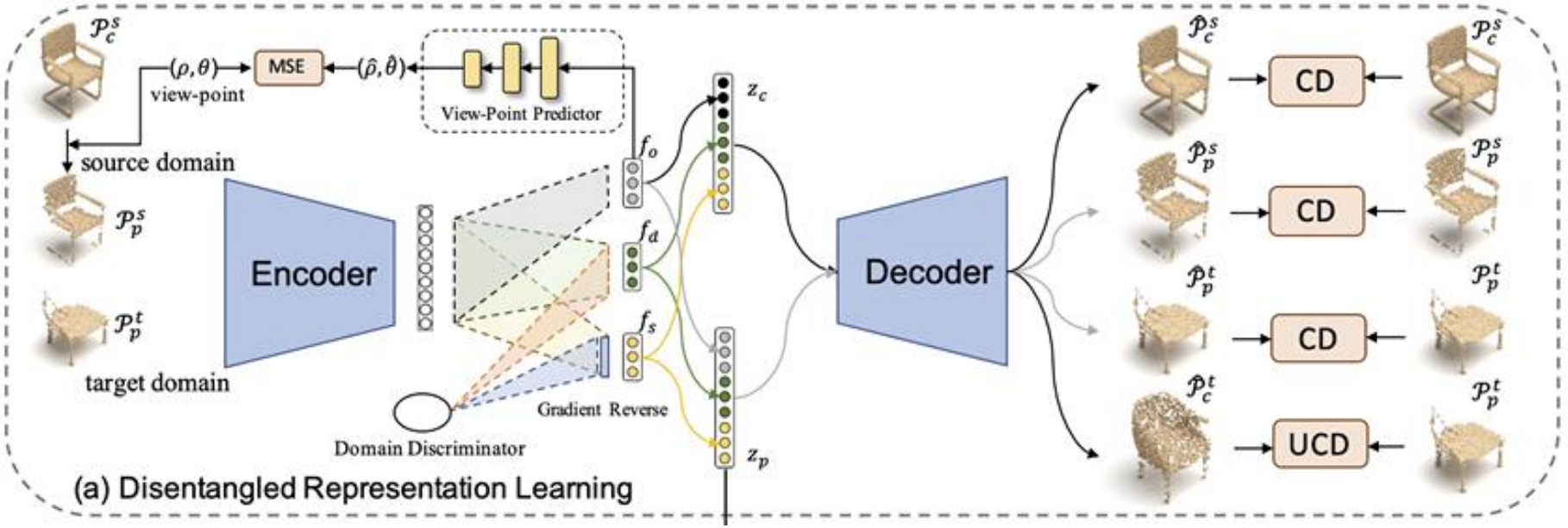
整体框架


Loss Function
Encoder
Decoder
Disentangler
Factor
Dicriminator Head
Gradient Reverse
MLPs
Zero Vector
source domain
target domain
view-point
MSE
View-Point Predictor
Encoder
Gradient Reverse
Domain Discriminator
Decoder
CD
CD
CD
UCD
(a) Disentangled Representation Learning
(b) Factor Permutation Consistency
exchange
Decoder
Encoder
MSE
MSE
(c) Optimization Stage
Initialization
MCD
Decoder
Input
Final Prediction

遮挡因子f_o自监督学习与视点预测

□ 建立“遮挡模式→视角”的映射：部分点云的缺失区域（遮挡模式）由扫描视角决定，因此若模型能从部分点云预测出扫描视角，说明它学习到了遮挡相关的特征。

□ 完整点云 → 视点采样 → 虚拟扫描 → 编码f_o→ 视点预测



遮挡因子f_o自监督学习与视点预测

自监督数据生成

1. 首先获取完整点云数据。
2. 对完整点云进行随机视角采样，其中视角的方位角ρ在0°到360°之间随机取值，仰角θ在-90°到90°之间随机取值。
3. 根据采样得到的视角参数，对完整点云进行旋转操作，得到旋转后的点云。
4. 运用Z-buffer遮挡渲染技术，对旋转点云进行渲染处理。
5. 通过渲染，得到部分点云P_p，该部分点云反映了特定视角下可见的点云信息。
6. 将采样得到的视角参数与渲染得到的部分点云P_p组合，构成自监督训练对，用于后续的自监督学习任务。

```
def partial_render_batch(pcl_batch, partial_batch, resolution=100, box_size=1.):
    batch_size, npoints, dimension = pcl_batch.shape
    rotmat_az_batch, rotmat_el_batch, az_batch, el_batch = generate_rotmat(batch_size)
    az_batch,el_batch = az_batch.reshape(batch_size,1), el_batch.reshape(batch_size,1)
    azel_batch = np.concatenate([az_batch,el_batch],1)
    point_visible_batch = ortho_render_batch(pcl_batch, rotmat_az_batch, rotmat_el_batch).reshape(batch
    for i in range(batch_size):
        point_visible = point_visible_batch[i,:,:]
        pcl = pcl_batch[i,:,:]
        point_visible_idx, _ = np.where(point_visible > 0.5)
        point_visible_idx = np.random.choice(point_visible_idx, 2048)
        new_pcl = pcl[point_visible_idx]
        partial_batch[i,:,:] = new_pcl
    return partial_batch, rotmat_az_batch, rotmat_el_batch, azel_batch
```

```
def ortho_render_batch(pcl_batch, rotmat_az_batch, rotmat_el_batch, resolution=100, npoints=2048, box_s
    pcl_batch = torch.matmul(pcl_batch, rotmat_az_batch)
    pcl_batch = torch.matmul(pcl_batch, rotmat_el_batch)

    depth = -box_size - pcl_batch[:,:,2]

    grid_idx = (pcl_batch[:,:,0:2] + box_size)/(2*box_size/resolution)

    plane_distance = torch.ones((batch_size*resolution*resolution*npoints)) * -box_size*2
    plane_distance[grid_idx] = depth

    point_visible = (plane_distance >= plane_depth)

    return point_visible.cpu().numpy()
```

遮挡因子f_o自监督学习与视点预测

解耦视角信息

1.PointNet++ 从部分点云提取包含遮挡信息的全局特征

2.解缠器Encoder从全局特征中分离出遮挡因子f_o

3.MLP 预测器根据f_o推断扫描视角，通过损失函数调整参数，迫使f_o编码视角相关的遮挡信息

```
self.DI_Disentangler = Disentangler(f_dims=96).cuda()
self.MS_Disentangler = Disentangler(f_dims=96).cuda()
self.DS_Disentangler = Disentangler(f_dims=96).cuda()
```

```
if self.args.vp_mode == 'matrix':
    virtual_f_vp = self.rotmatdecoder(virtual_f_vp)
    virtual_vp_loss = self.vp_criterion(virtual_f_vp, self.rotmat)
elif self.args.vp_mode == 'angle':
    virtual_vp_loss = self.vp_criterion(virtual_f_vp, self.azel)
else:
    raise NotImplementedError
di_loss_value += virtual_di_loss.item()
ds_loss_value += virtual_ds_loss.item()
vp_loss_value += virtual_vp_loss.item()
virtual_loss = (virtual_di_loss * 0.01 + virtual_ds_loss + virtual_vp_loss) * 0.004
virtual_loss.backward()
```

遮挡因子置零

$$z_c = 0 \otimes f_d \otimes f_s \rightarrow z_c = 0 \otimes f_d \otimes f_s$$

```
f_combine = torch.cat([f_di, f_ms, f_ds], 1)
f_combine_c = torch.cat([f_di, f_ms*0., f_ds], 1)
x_rec = self.Decoder(f_combine)
x = self.Decoder(f_combine_c)
```

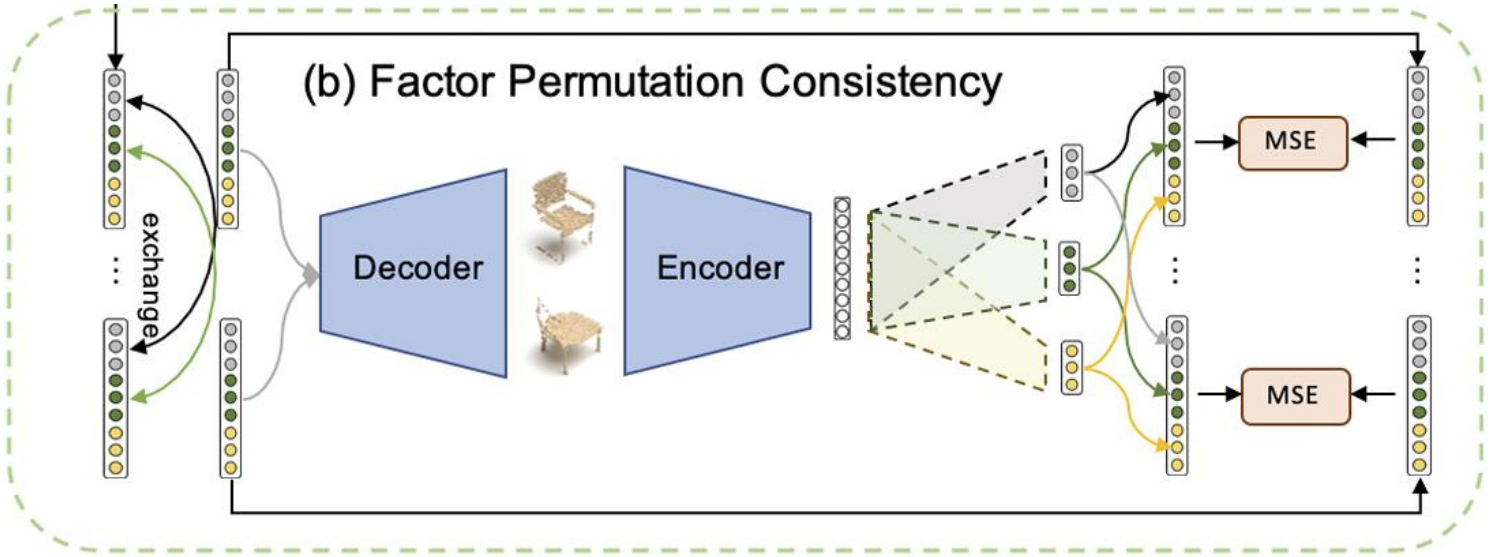
因子置换一致性

□ 核心挑战：在解耦表示中，需保证三个因子**完全独立**

- 独立性：修改任一因子时，其他因子不应发生系统性变化
- 若独立性不成立：补全时置零 f_o 可能意外改变物体形状或域属性（如将真实扫描椅补全成合成风格）

□ 解决思路：

通过**因子置换**+**重构一致性约束**强制因子独立



因子置换一致性具体步骤

□ 因子置换

做法：在同一个Batch内随机交换样本间的f_o或f_d，生成新组合因子 $\tilde{z}^i$：

$$\tilde{z}^i = f_o^j \otimes f_d^i \otimes f_s^i \text{ 或 } \tilde{z}^i = f_o^i \otimes f_d^j \otimes f_s^i$$

其中 j 是随机置换索引（如样本A的 f_o 替换样本B的 f_o，这个本质是将“椅子A的遮挡模式”与“椅子B的形状+域”组合，生成新虚拟样本

```
with torch.no_grad():
    cons_feature_di = torch.cat([virtual_f_di, real_f_di], 0)
    cons_feature_ds = torch.cat([virtual_f_ds, real_f_ds], 0)
    cons_feature_ms = torch.cat([virtual_f_ms, real_f_ms], 0)
    if switch_idx_default is None:
        switch_idx = np.random.randint(3)
    else:
        switch_idx = switch_idx_default
    switch_perm = np.random.permutation(cons_feature_di.shape[0])
    batch_size = cons_feature_di.shape[0]
    switch_perm = np.arange(batch_size)
    switch_perm = np.concatenate([switch_perm[batch_size//2:],switch_perm[:batch_size//2]], a
    if switch_idx == 0:
        pass
        cons_feature_di = cons_feature_di[switch_perm]
    elif switch_idx == 1:
        cons_feature_ms = cons_feature_ms[switch_perm]
    elif switch_idx == 2:
        cons_feature_ds = cons_feature_ds[switch_perm]
    cons_feature = torch.cat([cons_feature_di,cons_feature_ms, cons_feature_ds], 1)
```

1. 将虚拟数据和真实数据的域无关、缺失形状、域特定这三种因子分别拼接得到完整的特征
2. 随机生成0到2之间的整数作为switch_idx以表示要置换的因子类型
3. 获取拼接后特征的批量大小，生成从0到batch_size-1的整数数组并交换其前后半部分，形成置换索引
4. 依据switch_idx的值对相应的因子进行置换
5. 将置换后的三种因子在特征维度上拼接

因子置换一致性具体步骤

□ 解码器-编码器重构

前向过程：

- **输入：置换后的因子 $\tilde{z}^i$**
- **解码：生成点云 $\hat{p} = Dec(\tilde{z}^i)$**
- **期望：生成具有B的形状+域属性+A的遮挡模式的点云**
- **编码：将 $\hat{p}$ 重新编码回因子：**

$$\tilde{z}^i_{rec} = Enc(\hat{p}) = [f_o^{rec}, f_d^{rec}, f_s^{rec}]$$

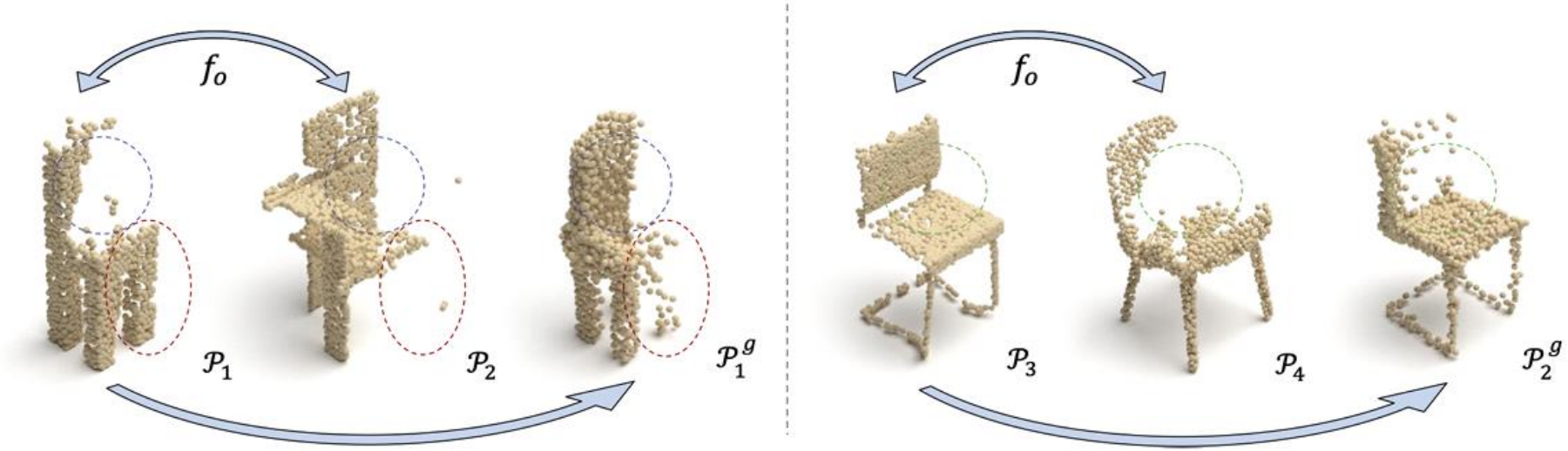
一致性约束：要求重构因子 $\tilde{z}^i_{rec}$ 与原始置换因子 $\tilde{z}^i$ 完全相同：

$$\mathcal{L}cons = \sum_{i=1}^{B} \left\| \tilde{z}^i_{rec} - \tilde{z}^i \right\|_2^2$$

```python
def train_consistency_one_batch(self, curr_step, cons_feature, return_generated=False, ith=-1):
    loss_dict = {}
    count = 0
    stage = 0
    consistency_loss_value = 0
    # setup learning rate
    for model_name in ["Encoder", "Decoder", "DI", "DS", "MS"]:
        self.schedulers[model_name].update(curr_step, self.args.G_lrs[0], ratio=0.99998)
    # forward
    for model_name in ["Encoder", "Decoder", "DI", "DS", "MS"]:
        self.models[model_name].eval()
    for model_name in ["Encoder", "Decoder", "DI", "DS", "MS"]:
        self.models[model_name].optim.zero_grad()
    x = self.Decoder(cons_feature)
    tree = [x]
    hidden_z = self.Encoder(tree)
    f_di = self.DI_Disentangler(hidden_z)
    f_ms = self.MS_Disentangler(hidden_z)
    f_ds = self.DS_Disentangler(hidden_z)
    f_combine = torch.cat([f_di, f_ms, f_ds], 1)
    consistency_loss = self.consistency_criterion(f_combine, cons_feature)
    consistency_loss_value += consistency_loss.item()
    consistency_loss *= 0.06
    consistency_loss.backward()
    for model_name in ["Encoder", "Decoder", "DI", "DS", "MS"]:
        self.models[model_name].optim.step()
    if return_generated:
        return x
    else:
        return consistency_loss_value
```

1. 使用组合特征生成点云
2. 将生成的点云重新编码，提取三个因子
3. 重新组合特征
4. 一致性约束：重构后的特征应与输入特征一致

因子置换一致性结果



1. **遮挡因子成功控制了扫描视角导致的缺失区域，且未干扰形状信息。**

2. **验证遮挡因子的独立性，即其仅影响 “哪里被遮挡”，而不影响 “物体是什么”。**

优化配适

输入：目标域部分点云。

具体步骤：

1. 编码器提取因子： $f_o \otimes f_d \otimes f_s = Enc(P)$

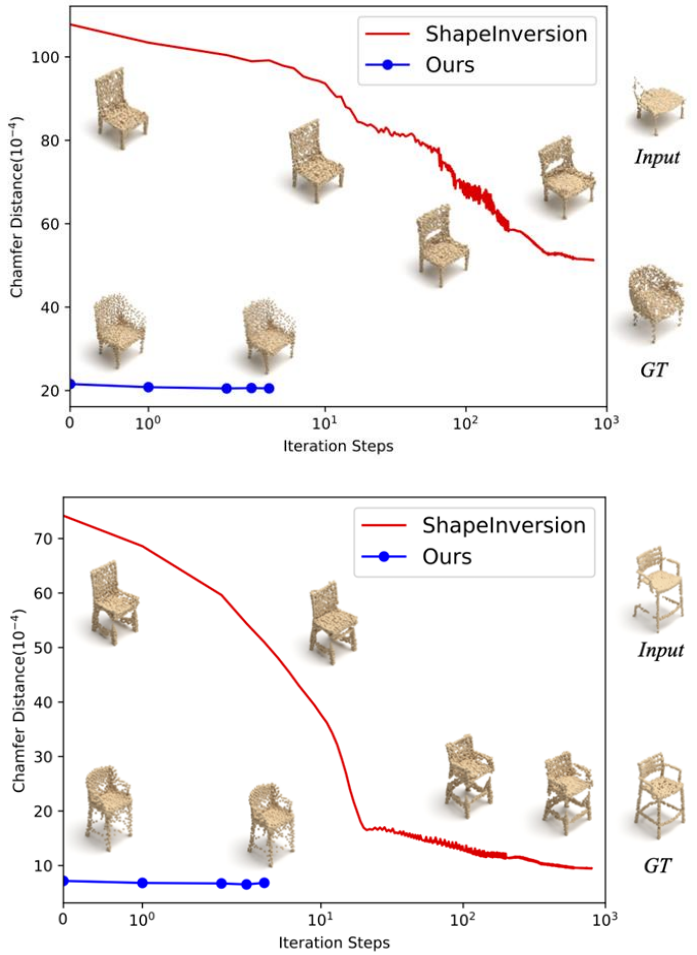
2. 初始化完整点云： $z_{init} = 0 \otimes f_d \otimes f_s$

3. 优化目标：最小化掩码Chamfer距离+潜变量正则化：

$$\min_{z,Dec} CD = (M(Dec(z)), p) \cdot \|z\|_2^2$$

$M(\cdot)$：**取预测点云中与输入部分最近邻的点集。**

优势：比纯优化方法（如ShapeInversion）快100倍



优化配适

编码器提取因子后初始化完整点云，计算掩码Chamfer距离

```python
def test_real_one_batch(self, bool_gt=False, use_ema=False, ith=-1):
    loss_dict = {}
    count = 0
    stage = 0

    # forward
    tree = [self.partial]
    for model_name in ["Encoder", "Decoder", "DI", "DS", "MS"]:
        self.models[model_name].eval()
    hidden_z = self.Encoder(tree)
    f_di = self.DI_Disentangler(hidden_z)
    #f_di_c = self.Z_Mapper(f_di)
    f_ms = self.MS_Disentangler(hidden_z)
    f_ds = self.DS_Disentangler(hidden_z)
    f_combine_c = torch.cat([f_di, f_ms*0., f_ds], 1)
    x = self.Decoder(f_combine_c)

    ### compute losses
    #ftr_loss = self.criterion(self.ftr_net, x_map, self.target)
    #ftr_loss = self.criterion(self.ftr_net, x, self.gt)

    #dist1, dist2 , _, _ = distChamfer(x_map, self.target)
    dist1, dist2 , _, _ = distChamfer(self.partial, x)
    directed_cd_loss = dist1.mean()
    if bool_gt:
        dist1, dist2 , _, _ = distChamfer(x, self.gt)
        cd_loss = dist1.mean() + dist2.mean()
```

渐进式优化
有GT: 0+0+2+1
无GT: 1+4+4+1，一共4阶段10迭代

```python
def finetune(self, bool_gt=False, save_curve=False, ith=-1):

        tree = self.z
        x = self.Decoder(tree)

        # masking
        x_map = self.pre_process(x,stage=stage)

        ### compute losses
        ftr_loss = self.criterion(self.ftr_net, x_map, self.partial)

        dist1, dist2 , _, _ = distChamfer(self.partial, x)
        ucd_loss = dist1.mean()
        dist1, dist2 , _, _ = distChamfer(x_map, self.partial)
        cd_loss = dist1.mean() + dist2.mean()
        if self.gt is not None:
            dist1, dist2 , _, _ = distChamfer(x, self.gt)
            gt_cd_loss = dist1.mean() + dist2.mean()
        # optional early stopping
        if self.args.early_stopping:
            if cd_loss.item() < self.args.stop_cd:
                break

        # nll corresponds to a negative log-likelihood loss
        nll = self.z**2 / 2
        nll = nll.mean()

        ### loss
        loss = ftr_loss * self.w_D_loss[0] + nll * self.args.w_nll \
            + cd_loss * 1
```

不会因为生成合理的补全部分而增加损失
确保已有部分的形状不会被破坏
给予模型在缺失部分生成时的自由度

```python
def k_mask(self, target, x, stage=-1):
    """
    masking based on CD.
    target: (1, N, 3), partial, can be < 2048, 2048, > 2048
    x: (1, 2048, 3)
    x_map: (1, N', 3), N' < 2048
    x_map: v1: 2048, 0 masked points
    """
    stage = max(0, stage)
    knn = self.args.k_mask_k[stage]
    if knn == 1:
        cd1, cd2, argmin1, argmin2 = distChamfer(target, x)
        idx = torch.unique(argmin1).type(torch.long)
    elif knn > 1:
        # dist_mat shape (B, 2048, 2048), where B = 1
        dist_mat = distChamfer_raw(target, x)
        # indices (B, 2048, k)
        val, indices = torch.topk(dist_mat, k=knn, dim=2,largest=False)
        # union of all the indices
        idx = torch.unique(indices).type(torch.long)

    if self.args.masking_option == 'element_product':
        mask_tensor = torch.zeros(2048,1)
        mask_tensor[idx] = 1
        mask_tensor = mask_tensor.cuda().unsqueeze(0)
        x_map = torch.mul(x, mask_tensor)
    elif self.args.masking_option == 'indexing':
        x_map = x[:, idx]

    return x_map
```

数据集

源域

CRN（源自ShapeNet）
数据：8个类别的30174个部分-完整点云对（每点云2048点）。
类别：飞机、汽车、椅子等（取6类与其他数据集重叠）。
生成方式：虚拟扫描合成部分点云（配对完整形状可用）。

监督信号来源	源域（CRN）	目标域（ScanNet等）
部分点云	✔	✔
完整点云GT	✔	✘
自监督任务	✔	✔

目标域

真实扫描数据集（无完整GT）
1. ScanNet：室内场景扫描（桌椅类别）。
2. MatterPort3D：室内场景扫描（桌椅类别）。
3. KITTI：室外自动驾驶扫描（汽车类别）。
预处理：重采样至 2048点 （与源域对齐）。

合成数据集（有完整GT）
1. 3D-FUTURE：
高真实感家具模型（柜子、椅子、灯、沙发、桌子）。
生成方式：从5个固定视角渲染部分点云，完整点云通过表面采样。
2. ModelNet（ModelNet40子集）：
生成方式：虚拟扫描生成部分点云，表面采样生成完整点云（各2048点）。

指标测试

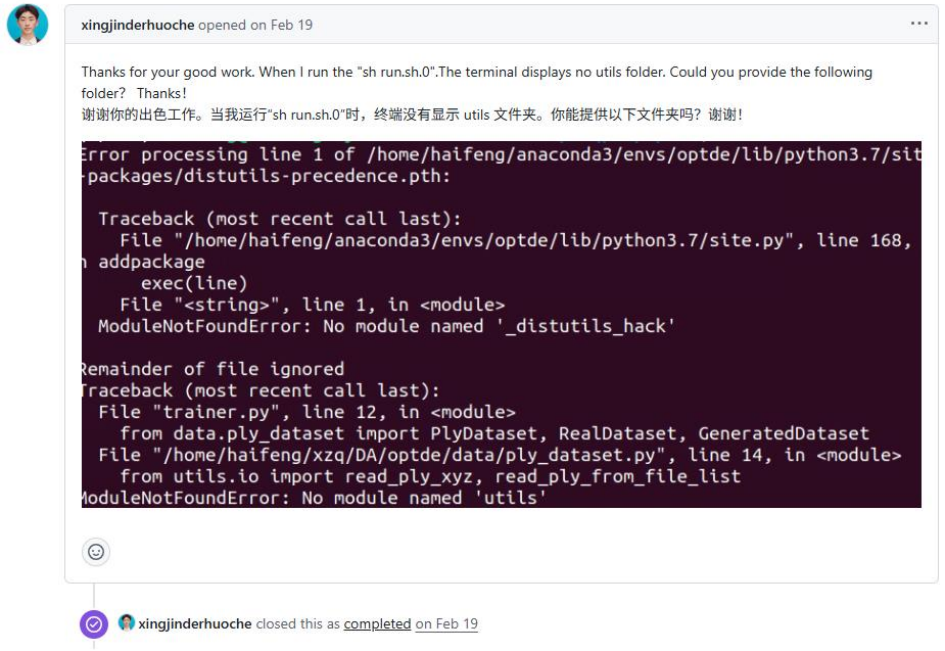
无GT时，评测的是补全的**局部合理性**（输入是否被保留）
有GT时，评测的是补全的**全局真实性**（预测是否像真物体）

指标	数据要求	评估目标	直观解释	适用场景
UCD	仅需输入部分点云	输入部分与预测完整点云的覆盖度	输入点到预测点的平均最小距离 → 衡量“预测是否覆盖输入结构”	真实扫描（ScanNet/KITTI等无GT数据）
UHD	仅需输入部分点云	输入与预测的最差局部对齐误差	输入点中离预测点最远的距离 → 检测“是否存在严重偏离点”	需评估异常值（如噪声点）的场景
CD	需输入部分+完整GT	预测与GT的全局形状一致性	双向最近邻距离均值 → 同时衡量“预测多像GT”和“GT多像预测”	合成数据（3D-FUTURE/ModelNet有GT时）

复现过程遇到的问题

About utils folder #3
关于 utils 文件夹 #3

Closed



xingjinderhuoche opened on Feb 19

Thanks for your good work. When I run the "sh run.sh.0".The terminal displays no utils folder. Could you provide the following folder? Thanks!
谢谢你的出色工作。当我运行"sh run.sh.0"时，终端没有显示 utils 文件夹。你能提供以下文件夹吗？谢谢！

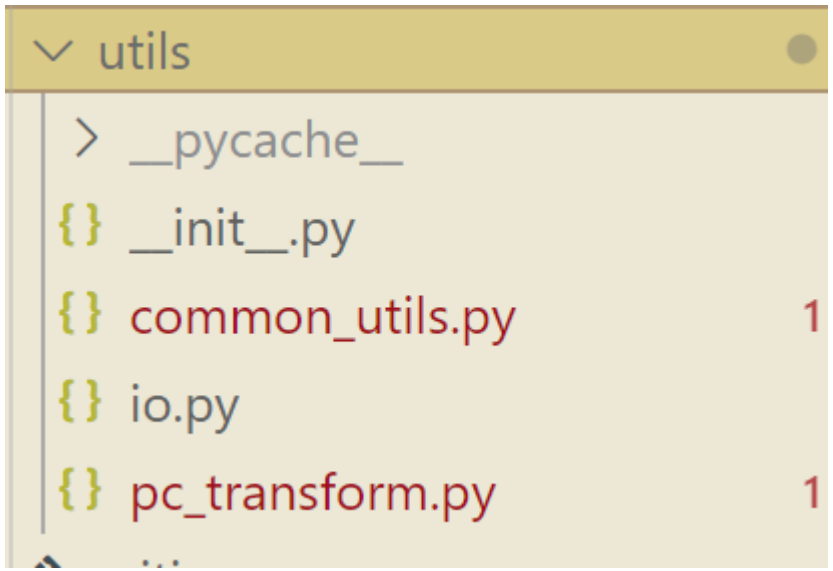
```
Error processing line 1 of /home/haifeng/anaconda3/envs/optde/lib/python3.7/sit
-packages/distutils-precedence.pth:

  Traceback (most recent call last):
    File "/home/haifeng/anaconda3/envs/optde/lib/python3.7/site.py", line 168,
n addpackage
      exec(line)
    File "<string>", line 1, in <module>
  ModuleNotFoundError: No module named '_distutils_hack'

Remainder of file ignored
Traceback (most recent call last):
  File "trainer.py", line 12, in <module>
    from data.ply_dataset import PlyDataset, RealDataset, GeneratedDataset
  File "/home/haifeng/xzq/DA/optde/data/ply_dataset.py", line 14, in <module>
    from utils.io import read_ply_xyz, read_ply_from_file_list
ModuleNotFoundError: No module named 'utils'
```

xingjinderhuoche closed this as completed on Feb 19

解决方案：根据导入的代码构造utils下面的python文件



代码训练

run.sh配置

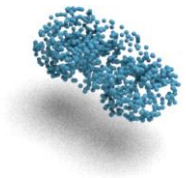
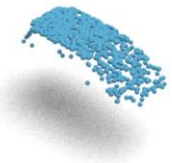
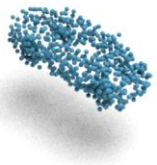
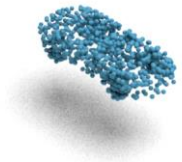
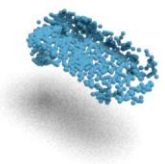
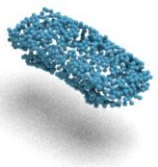
```
VIRTUALDATA=CRN
REALDATA=ModelNet
VCLASS=car
RCLASS=car
LOGDIR=logs
CUDA_VISIBLE_DEVICES=$1 python trainer.py \
--virtualdataset ${VIRTUALDATA} \
--realdataset ${REALDATA} \
--class_choice ${VCLASS} \
--split train \
--epoch 200 \
--mask_type k_mask \
--save_inversion_path ./${LOGDIR}/${REALDATA}_${RCLASS} \
--ckpt_load pretrained_models/${VCLASS}.pt \
--dataset_path ./datasets/our_data/ \
--log_dir ${LOGDIR}
```

训练日志

```
1  ##########EPOCH 0000##########
2  Mean Chamfer Distance on Training Set:0.00270367
3  Mean UCD on Real Training Set:0.00156744
4  Mean Loss on Training Set DI:1.38803978DS:1.39990767VP:6.65625824CS:0.00000000
5  Mean Chamfer Distance on Test Set:0.00113760
6  Mean CD on Real Test Set:0.00126408
7  Mean UCD on Real Test Set:0.00081907
8  Mean UHD on Real Test Set:0.06753373
9  ##########EPOCH 0001##########
10 Mean Chamfer Distance on Training Set:0.00125057
11 Mean UCD on Real Training Set:0.00185146
12 Mean Loss on Training Set DI:1.38644586DS:1.39395840VP:2.92401503CS:0.00000000
13 Mean Chamfer Distance on Test Set:0.00129244
14 Mean CD on Real Test Set:0.00113763
15 Mean UCD on Real Test Set:0.00059116
16 Mean UHD on Real Test Set:0.05548330
17 ##########EPOCH 0002##########
18 Mean Chamfer Distance on Training Set:0.00119514
19 Mean UCD on Real Training Set:0.00129554
20 Mean Loss on Training Set DI:1.38630145DS:1.38988301VP:2.10959676CS:0.00000000
21 Mean Chamfer Distance on Test Set:0.00113538
22 Mean CD on Real Test Set:0.00110344
23 Mean UCD on Real Test Set:0.00058561
24 Mean UHD on Real Test Set:0.05707238
25 ##########EPOCH 0003##########
```

可视化结果

以ModelNet为例训练后的可视化结果

groundtruth　　input(partial)　　output

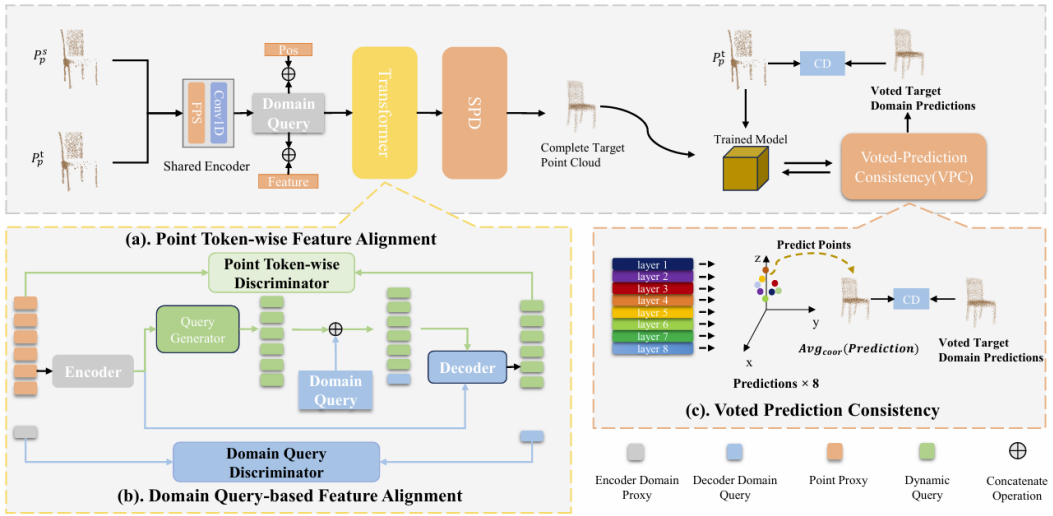
指标测试

以ModelNet为例训练后的CD结果


```
{
  "mean_chamfer_distance": 0.0017323221258828704,
  "std_chamfer_distance": 0.0007650791891618531,
  "min_chamfer_distance": 0.0006195167079567909,
  "max_chamfer_distance": 0.005974980536848307,
  "num_samples": 485,
```

改进点：后续文章的分析

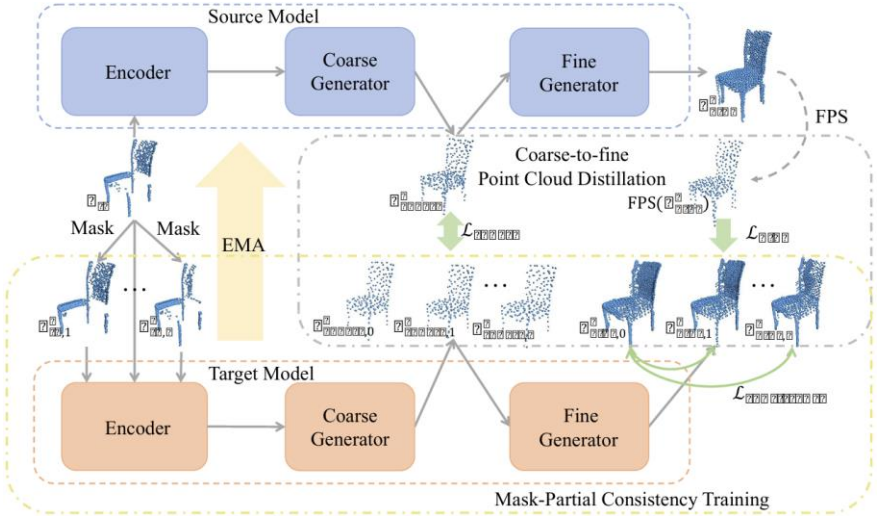
[AAAI 2025] DAPoinTr: Domain Adaptive Point Transformer for Point Cloud Completion



关键思想：

- 解决了**OptDE在源数据获取方面的限制**，使其更适用于源数据不可得的实际场景。
- PointSFDA 是首个针对点云补全的**无源域适应**（SFDA）框架，它仅**依靠预训练的源模型和未标记的目标数据**进行适应，**无需访问源域数据**。

[ARXIV 2025] PointSFDA: Source-free Domain Adaptation for Point Cloud Completion

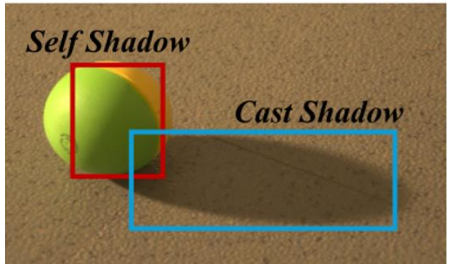


关键思想：

- OptDE 在处理目标域的局部几何信息时，由于**域间隙可能引入噪声**，导致**局部细节的补全效果不够理想**
- 提出了自监督的部分掩码一致性训练策略。该策略通过对**输入点云进行不同的掩码增强**，并**确保不同掩码下的点级预测一致性**，从而使模型能够学习到目标域的局部几何信息

改进点：个人从方法以及任务本身的思考

Q1：遮挡因子貌似不能解决非规则遮挡（如物体自身复杂结构导致的自遮挡）问题，比如某个镂空的物体，很难用遮挡因子来处理



Q2：目前的方法的motivation是基于类别已知，现有的方案依赖预定义类别划分，如果在实际场景中出现未知类别物体的实时扫描，那么缺乏对开放集场景的适应性。这个是不是可以往开放世界（Open World）延伸?