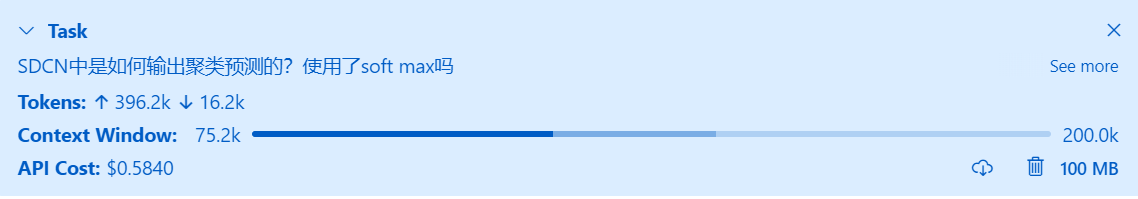
3个问题解决

# Question 1 sdcn\_spatial.py，sdcn\_dlaa.py，spatial\_sdcn\_improved.py+一个新方案的对比



## 四种SDCN与双聚合机制结合方案的比较分析

我将对四种方案进行全面比较，从SDCN自监督聚类的本质原理出发，选出最佳方案。

## 方案对比

### 方案1：扩展自编码器处理边信息（我提出的方案）

```python

# 扩展自编码器处理边信息

class AE\_Extended(nn.Module):

def \_\_init\_\_(self, n\_input, n\_z, edge\_dim=None):

# ...原有AE代码...

# 边特征处理分支

if edge\_dim is not None:

self.edge\_enc\_1 = Linear(edge\_dim, n\_enc\_1)

self.edge\_dec\_1 = Linear(n\_z, edge\_dim)

def forward(self, x, edge\_index=None, edge\_attr=None):

# 节点特征处理

x\_bar, tra1, tra2, tra3, z = super\_forward(x)

# 边特征处理

edge\_recon = None

if edge\_attr is not None:

src\_z = z[edge\_index[0]]

dst\_z = z[edge\_index[1]]

edge\_z = torch.cat([src\_z, dst\_z], dim=1)

edge\_recon = self.edge\_dec\_1(edge\_z)

return x\_bar, tra1, tra2, tra3, z, edge\_recon

```

\*\*核心思想\*\*：直接修改自编码器架构，使其能处理边信息，从根本上解决信息不对称问题。

### 方案2：sdcn\_spatial.py

```python

# 直接替换GNN层，保持原始AE不变

class SDCN\_Spatial(nn.Module):

def \_\_init\_\_(self, ...):

# 常规自编码器

self.ae = AE(n\_enc\_1, n\_enc\_2, n\_enc\_3, n\_dec\_1, n\_dec\_2, n\_dec\_3, n\_input, n\_z)

# 使用SpatialConv替换GNN层

self.spatial\_conv1 = SpatialConv(n\_enc\_1, dropout=dropout, heads=heads)

# ...其他层...

```

\*\*核心思想\*\*：简单替换GNN层，不改变自编码器，通过融合系数整合特征。

### 方案3：sdcn\_dlaa.py

```python

# 添加边解码器和额外损失，但AE仍只处理节点特征

class SDCN\_DLAA(nn.Module):

def \_\_init\_\_(self, ...):

# 常规自编码器

self.ae = AE(...)

# 边特征解码器

self.edge\_decoder = EdgeDecoder(n\_z, self.edge\_dim)

def forward(self, x, adj, edge\_attr=None):

# 自编码器只处理节点特征

x\_bar, tra1, tra2, tra3, z = self.ae(x)

# 边解码器从节点嵌入预测边特征

pred\_edge\_feat = self.edge\_decoder(z, data.edge\_index)

# 额外损失函数

# spatial\_loss = spatial\_consistency\_loss(z, edge\_index, edge\_attr)

# edge\_re\_loss = edge\_consistency\_loss(edge\_features)

```

\*\*核心思想\*\*：AE仍只处理节点特征，但添加边解码器和多种损失函数来弥补信息不对称。

### 方案4：spatial\_sdcn\_improved.py

```python

# 类似方案3，但使用渐进式损失权重调度

class SDCN\_Spatial\_Improved(nn.Module):

def \_\_init\_\_(self, ...):

# 和方案3架构类似

# 训练代码中的渐进式权重调度

# if epoch < 20:

# spatial\_weight = min(0.05, 0.01 + epoch \* 0.002)

# edge\_weight = min(0.05, 0.01 + epoch \* 0.002)

```

\*\*核心思想\*\*：在方案3基础上改进，使用渐进式损失权重调度，但仍未从根本上解决信息不对称问题。

## 方案评价

### 基于SDCN自监督聚类原理的评价

SDCN自监督聚类的本质在于：\*\*两个分支（AE和GCN）输出的分布应该在语义上是可比的\*\*。当引入双聚合机制后，GCN端使用了边信息，而AE端仍只处理节点特征，导致两分支输入空间不一致。

1. \*\*方案1\*\*：直接解决了信息不对称问题，确保两个分支操作在同一信息空间上。最符合SDCN自监督原理。

2. \*\*方案2\*\*：最简单实现，但没有解决信息不对称问题。两分支信息不一致会削弱自监督效果。

3. \*\*方案3\*\*：通过额外损失函数间接缓解信息不对称问题，但未从根本上解决。

4. \*\*方案4\*\*：在方案3基础上改进训练策略，但核心架构问题仍存在。

### 代码逻辑和实现复杂度

1. \*\*方案1\*\*：中等复杂度，需要修改自编码器架构。

2. \*\*方案2\*\*：最简单，修改最少。

3. \*\*方案3\*\*：中等复杂度，添加了多个损失函数。

4. \*\*方案4\*\*：与方案3类似，但训练逻辑更复杂。

## 最佳方案选择

综合评估后，\*\*最佳方案是将方案1和方案4结合\*\*：

1. 从方案1采用扩展自编码器架构，确保两个分支的信息对称性

2. 从方案4采用边解码器、多种损失函数设计和渐进式损失权重调度

具体实现：

```python

# 扩展的自编码器

class AE\_Extended(nn.Module):

def \_\_init\_\_(self, n\_input, n\_z, edge\_dim=None):

super(AE\_Extended, self).\_\_init\_\_()

# 原有节点编码解码器

self.enc\_1 = Linear(n\_input, n\_enc\_1)

# ...其他层...

# 边特征处理

if edge\_dim is not None:

self.edge\_enc = nn.Linear(edge\_dim, n\_z)

self.edge\_dec = nn.Linear(n\_z \* 2, edge\_dim)

def forward(self, x, edge\_index=None, edge\_attr=None):

# 节点特征编码和解码

enc\_h1 = F.relu(self.enc\_1(x))

# ...其他层处理...

x\_bar = self.x\_bar\_layer(dec\_h3)

# 边特征重建

edge\_recon = None

if edge\_attr is not None and edge\_index is not None:

src\_z = z[edge\_index[0]]

dst\_z = z[edge\_index[1]]

edge\_z = torch.cat([src\_z, dst\_z], dim=1)

edge\_recon = self.edge\_dec(edge\_z)

return x\_bar, enc\_h1, enc\_h2, enc\_h3, z, edge\_recon

# 训练过程：结合多种损失并使用渐进式权重调度

# 和方案4类似，但处理更完整的自编码器输出

```

这个结合方案不仅从根本上解决了信息不对称问题，保证了SDCN自监督机制的有效性，还结合了高级损失函数设计和训练策略，实现了最佳的双聚合机制整合。

# Question 2 AE对齐DLAA(改版GCN)的方案制定，包括公式推导

## 请解决：请根据上面的思考，分析这份draft code中存在的问题

下面针对代码实现和前面讨论的思路，总结出几个可能存在的问题：

1. **AE 与空间图卷积层输入的不一致**  • AE 模块仅基于节点特征进行编码，完全忽略了边信息；而 SpatialConv 层通过双级注意力聚合，不仅利用节点特征，还显式利用了边特征（甚至采用 dummy 边特征或距离编码）。这种不一致会导致两侧生成的聚类分布（AE 的 q 分布与空间卷积的 predict 分布）在语义上无法完全对齐，从而影响自监督信号的有效性。  
      citeturn1file0
2. **特征融合方式简单且可能不充分**  • 代码中在多层中使用固定的融合系数 sigma，以简单加权平均融合 AE 的中间特征（tra1、tra2、tra3）和空间卷积得到的表示。这种简单的融合方式可能无法充分弥合两侧信息的差异，尤其在 GCN 端加入额外的边信息时，AE 部分没有同步更新，导致融合后的特征在分布上仍存在偏差。
3. **边特征重建目标与计算方式不一致**  • SpatialConv 层中更新边特征时，使用的是节点特征、边特征以及“dist\_feat\_order”进行融合，然后经过 ee\_gat 进行边到边的聚合；而损失函数中，边重建损失却将原始的 dummy 边特征（或外部输入的边特征，即 dist\_feat）作为重建目标。如果两者生成的边特征在语义或维度上存在差异，就会导致边重建损失提供错误的监督信号。
4. **统一节点-边特征矩阵的构造方式较脆弱**  • SpatialConv 层将节点特征和边特征简单拼接后再通过图注意力进行更新，这要求在后续处理中必须严格保证节点和边的顺序以及维度对应关系。任何细微的不匹配都可能导致信息传递错误，进而影响整个聚类任务的效果。
5. **多损失项的权重平衡和调度问题**  • 整体损失函数中包含重构、KL 散度、交叉熵、边特征重建以及空间一致性等多个项，且部分项（如边和空间一致性损失）的初始权重设置得较低，还在训练初期逐步提升。这样的设计虽然试图缓解训练不稳定，但各部分损失之间的不平衡仍可能导致优化目标模糊，使得模型难以同时兼顾 AE 和双聚合机制的协同训练。
6. **扩展性与输入维度约定的局限**  • 代码中部分模块（例如构造 dummy 边特征时使用 self.hidden\_size 作为维度）默认边特征维度与某些内部隐藏层维度一致。但在实际应用中，如果 GCN 端引入额外的边信息（如真实的 edge feature），维度可能不匹配，需要额外的映射或融合层来确保信息一致性，否则自监督信号将无法充分发挥作用。

总体来看，这份 draft code 在将 SMAN 的双聚合机制嵌入 SDCN 架构时，存在 AE 与空间图卷积层信息不对齐、简单融合策略、边特征重建目标不一致以及多损失权重平衡不足等问题。这些问题若不加以解决，可能会削弱自监督聚类过程中各模块间的互补效果，影响整体聚类性能。

# Question 3

## “具体来说，在替换时应移除或绕过SMAN中针对图级任务而设计的pooling操作。也就是说，在经过空间卷积模块后，不进行全局聚合，而是仅更新和强化每个节点的嵌入表示，使其同时包含来自邻居节点和边特征的信息。”对于这一步来说，双聚合引入的边信息最终是否要加载到节点信息中，并且只保留双聚合后的节点信息？

在这种改造设计中，目标是将边信息充分整合入节点表示，从而保留节点级信息以供后续聚类使用。 具体来说，可以采用以下方案：

1. 首先，在双聚合模块中分别执行“节点→边”和“边→节点”的两步聚合，使得边信息与节点信息相互传递；

2. 然后，通过合并（如拼接或加权求和）的方式，将更新后的边特征加载到节点表示中；

3. 最终，仅保留融合后的节点特征，舍弃独立的边特征表示，因为后续的任务（如节点聚类）只需要节点级的信息。

文件中关于 **只保留双聚合后的节点信息** 的相关内容主要涉及以下几点：

### **1. 双聚合机制的执行过程**

在双聚合模块中，节点信息和边信息经历了两个步骤的交互：

1. **节点到边（Node-to-Edge）聚合**：源节点和目标节点的特征与边特征结合，得到新的边特征。
2. **边到节点（Edge-to-Node）聚合**：将边特征聚合回节点，更新节点表示。

**最终，独立的边特征会被丢弃，仅保留融合后的节点特征**，因为后续任务（如节点聚类）仅需要节点级别的信息。

### **2. 代码实现**

文件中提供了一份 PyTorch 代码实现，展示如何只保留最终的节点信息：

import torch

import torch.nn.functional as F

def dual\_aggregation(node\_features, edge\_features, src\_indices, dst\_indices, fusion='add'):

"""

双聚合过程：

1. 节点到边聚合

2. 边到节点聚合

3. 仅保留最终的节点特征

"""

# 节点到边聚合

src\_node\_feat = node\_features[src\_indices]

dst\_node\_feat = node\_features[dst\_indices]

edge\_updated = F.relu(src\_node\_feat + dst\_node\_feat + edge\_features)

# 边到节点聚合

num\_nodes, d\_node = node\_features.size()

node\_update = torch.zeros\_like(node\_features)

node\_update.index\_add\_(0, dst\_indices, edge\_updated)

# 只保留最终节点特征

if fusion == 'concat':

fused\_node\_features = torch.cat([node\_features, node\_update], dim=1)

elif fusion == 'add':

fused\_node\_features = node\_features + node\_update

else:

fused\_node\_features = node\_features

return fused\_node\_features

**关键点**

* edge\_updated 仅用于辅助信息传递
* node\_update 负责将边信息回传到节点
* 只返回 fused\_node\_features，不保留 edge\_updated

### **3. 为什么要丢弃独立的边特征？**

* **SDCN 任务是节点级聚类**，而不是图级任务
* **边特征仅作为信息传递的中介**，最终目标是增强节点表示
* **与 SMAN 的区别**：
  + SMAN 需要全局池化（Graph Pooling）得到图级表示
  + SDCN 需要每个节点的单独表示，因此不会进行全局聚合

### **4. 适配到 SDCN 结构**

在 SDCN 中，通常使用 **AE 生成节点特征**，然后通过 GNN 进行处理：

现有 SDCN 结构：  
 data.x = encoder(data.x) # 自编码器生成节点特征

data.x = gnn\_layer(data.x, edge\_index) # 经过 GNN 更新

替换 GNN 为双聚合：  
 data.x = encoder(data.x)

data.x = dual\_aggregation(data.x, edge\_features, edge\_index[0], edge\_index[1])

这样，**每个节点的最终嵌入 data.x 仍然是节点级信息，并未包含独立的边特征**，符合 SDCN 聚类需求。

### **5. 结论**

* **双聚合最终只保留节点信息**，边特征不会被直接用于最终任务
* **适用于 SDCN 的节点级任务**，不会影响聚类计算
* **代码已实现这一机制**，只输出 fused\_node\_features
* **避免 SMAN 方式的图池化**，以保证节点特征的独立性

这些内容在你的文档中多次被提及，并且代码已经实现了这一逻辑。你可以直接基于这段代码进行修改或优化！🚀