基于 UMAP–GNN 概率图对齐的分子三维构象生成

关键词：UMAP 核、行自适应核、GNN、概率图对齐、交叉熵、摊销推理、非参数微调

摘要

我们提出一种基于**概率图对齐**的分子构象生成方法。用真实构象 Y^{*} 经 **UMAP 低维核**构造**老师图** Q^{*}；用 **GNN** 从分子图预测**学生图** $\hat{P}$（两种核形式：固定 UMAP 同族核/行自适应核），以**交叉熵** $\mathrm{CE}(Q^{*}, \hat{P})$ 进行监督训练。推理时，给定新分子，GNN 预测 $\hat{P}$，再最小化 $\mathrm{CE}(\hat{P}, Q(Y))$ 获得 3D 坐标 Y。该框架天然刚体不变，具备"拉/推"双向信号，统一了摊销学习与非参数几何微调。

---

1. 背景与动机

- 传统路线：
  1. 等变 GNN/扩散直接回归/采样坐标；
  2. 距离几何/能量最小化；
  3. 统计嵌入 + 力场混合。
- 我们的视角：**流形学习**。把 3D 几何表述为**概率图**，将"化学表征流形"与"欧氏 3D 流形"做**概率对齐**，以 CE 为核心目标，稳定且可解释。

---

2. 老师图 Q^{*}：由真实坐标构建

给定真实/参考坐标 $Y^{*} = \{y_i^{*} \in \mathbb{R}^3\}$，用 **UMAP 低维核**（与 `min_dist, spread` 对应的 (a, b)）定义

$$Q_{ij}^{*} = \frac{1}{1 + a \, \|y_i^{*} - y_j^{*}\|^{2b}}, \qquad Q_{ii}^{*} = 0.$$

> 实践：将 Q^{*} 裁剪到 $[\varepsilon, 1-\varepsilon]$（如 10^{-12}），并在**构建数据集阶段离线缓存**。

---

3. 学生图 $\hat{P}$：由 GNN 学习（两种核）

3.1 路线 A（固定核，度量学习，推荐起步）

GNN 输出节点嵌入 $z_i \in \mathbb{R}^p$，两两距离

$$d_{ij} = \|z_i - z_j\|.$$

用与 $Q^{\setminus *}$ **同族的 UMAP 低维核**得到

$$\hat{P}_{ij} \;=\; \frac{1}{1 + a\, d_{ij}^{2b}}, \qquad \hat{P}_{ii} = 0.$$

优点：核族一致，CE 优化稳定；$\hat{P}$ 可**解析反解**为距离矩阵，便于 MDS 初始化 Y。

3.2 路线 B（行自适应核 + Fuzzy Union，UMAP 高维风格）

GNN 额外输出逐节点参数 $\rho_i, \sigma_i > 0$，并在 z-空间计算 d_{ij}。推荐两种行核：

- **指数行核**（UMAP 原始高维核）

$$q_{ij}^{(i)} \;=\; \exp\!\Big(-\frac{\max(0,\, d_{ij} - \rho_i)}{\sigma_i}\Big).$$

- **UMAP 同族行核**（与低维核同形）

$$q_{ij}^{(i)} \;=\; \frac{1}{1 + a\,\Big(\max\big(0, \tfrac{d_{ij}-\rho_i}{\sigma_i}\big)\Big)^{2b}}.$$

行对称化用 **fuzzy union**：

$$\hat{P}_{ij} \;=\; q_{ij}^{(i)} + q_{ji}^{(j)} - q_{ij}^{(i)}\, q_{ji}^{(j)}, \qquad \hat{P}_{ii} = 0.$$

优点：**密度自适应**，跨分子分布差异鲁棒；**缺点**：$\hat{P}$ 不可直接解析反解为欧氏距离（推理用 CE 对 Y 微调即可）。

---

4. 训练目标与正则

主目标（只在非对角上求和）：

$$\mathcal{L}_{\text{main}} = \frac{1}{|S|} \sum_{(i,j)\in S} \Big[- Q_{ij}^{\setminus *} \log \hat{P}_{ij} - (1 - Q_{ij}^{\setminus *}) \log(1 - \hat{P}_{ij}) \Big],$$

其中 $S = \{(i,j) \mid i \neq j\}$（可加 hop 掩码/负采样）。

建议正则：

- **Smooth-k 行强度（行自适应核时）** 令 $s_i = \sum_j q_{ij}^{(i)}$，目标 $s_i \approx \log_2(k_i + 1)$：

$$\mathcal{R}_k = \sum_i \big(s_i - \log_2(k_i + 1)\big)^2.$$

- **温度/范数**：约束 $\|z_i\|$ 或 $\hat{P}$ 的行熵，避免过稀疏/过稠密。
- **掩码与采样**：在 hop$\le K$ 上密集监督，同时对远程对进行随机负采样，控制复杂度。

总损失：

$$\mathcal{L} = \mathcal{L}_{\text{main}} \; + \; \lambda_k \, \mathcal{R}_k \; (+ \text{其他正则}).$$

---

5. 训练与数据缓存

- **数据端**：小分子建议**离线缓存** $Q^{\setminus *}$（或 $D^{\setminus *}$ 以便后续换核）；对角置 0，概率裁剪到 $(\varepsilon, 1-\varepsilon)$。
- **复杂度**：全对为 $O(N^2)$。对较大分子，使用 hop 掩码 + 负采样，近似 $O(NK)$。
- **稳定性**：概率裁剪、忽略对角、梯度裁剪；行核在 $d \le \rho_i$ 时导数为 0；候选极少的行可冻结。

---

6. 推理与几何解码（$\hat{P} \Rightarrow Y$）

通用（两条路线都适用）：

1. **GNN 前向**得到 $\hat{P}$；
2. **非参数 CE 微调**：以 $\hat{P}$ 为目标，最小化

$$\min_{Y} \; \mathrm{CE}(\hat{P}, Q(Y)), \quad Q_{ij}(Y) = \frac{1}{1 + a \, \|y_i - y_j\|^{2b}}.$$

   迭代 10–50 步即可收敛（刚体不变，数值稳）。

路线 A 可加速（固定核可反解）：

$$\hat{D}_{ij} \; = \; \left(\frac{1/\hat{P}_{ij} - 1}{a} \right)^{\frac{1}{2b}},$$

用 $\hat{D}$ 先做 **MDS/SMACOF** 得到初值 Y_0，再做少量 CE 微调到 Y。

---

7. 模型结构（建议）

- **Encoder**：原子/键特征 $\to$ 多层 GNN（GINE/MPNN/GraphTransformer）$\to h_i$。
- **Metric 头（路线 A）**：$z_i = \mathrm{MLP}(h_i)$；按 UMAP 低维核算 $\hat{P}$。
- **Row-adaptive 头（路线 B）**：
  $\rho_i = \mathrm{softplus}(\mathrm{MLP}(h_i)) + \epsilon$，$\sigma_i = \mathrm{softplus}(\mathrm{MLP}(h_i)) + \epsilon$；行核 + fuzzy union 得 $\hat{P}$。
- **可选多任务**：行强度/温度预测、hop 可信度等。

---

8. 评测与基线

- **指标**： RMSD（Kabsch 对齐）、键长/键角/1-4 距离误差、立体冲突计数、MMFF 能量（参考）、$\mathrm{CE}(Q^{\setminus *}, Q(Y))$。
- **基线**： ETKDG(+优化)、距离几何/MDS-stress、等变扩散/EGNN 坐标回归、无核对齐的对比学习。
- **消融**： 去掉 smooth-k、去掉 hop 掩码、核族切换（exp vs UMAP 同族）、仅正项 CE、仅局部监督等。

---

9. 创新点（可写入论文贡献）

1. **目标空间创新**：主目标不是能量/坐标回归，而是**概率图 CE 对齐**（含推开项）。天然刚体不变、数值稳定，这和主流的 E(n)-GNN/扩散（坐标生成）是不同的范式。
2. **统一监督/无监督**：同一 CE 框架下，teacher 可取 $Q^{\setminus *}(Y^{\setminus *})$（监督）或表征图 P（无监督）。这在分子 3D 里很少见。
3. **行自适应核 + fuzzy union** 与 **固定 UMAP 低维核**两条学生图：一个密度鲁棒、一个可解析反解（利于初始化），**工程上好用**。
4. **摊销 + 非参数微调**：先用 GNN 预测 $\hat{P}$，再少量步 CE 微调 Y。相比一次性重构 Y，**推理速度/稳健性**兼顾。
5. **可辨识性切入点**：当 a, b 固定、边覆盖足够、CE→0 时，得到的 Y 与 $Y^{\setminus *}$ 等距同构（至多刚体/镜像），这提供了**理论抓手**（现有扩散/能量方法很少从概率图角度给出）。

---

10. 高影响力"杀手锏"实验（建议优先做）

- **Teacher 噪声鲁棒**：对 $Y^{\setminus *}$ 加噪生成 $Q^{\setminus *}$（或只给局部对），看你方法的收敛与恢复能力。
- **速度–质量曲线**：在相同 wall-clock 下，你的"摊销 $\hat{P}$ + CE 微调"对比扩散/能量最小化。
- **跨分子泛化**：训练分布外化学家族（含多环/芳杂环/卤素/高自由度侧链），报告稳定性。
- **解析反解 + SMACOF 的加速收益**（route-A），以及 **不反解直接 CE 微调**（route-B）的对比。
- **局部监督→全局几何**：只给 hop≤2 的 $Q^{\setminus *}$，看是否能重建全局（用刚性理论做讨论）。

11.题目/卖点示例

- **"Probabilistic Manifold Alignment for Molecular Conformer Generation via UMAP Kernels"** *Aligning Teacher–Student Probability Graphs with Cross-Entropy to Recover 3D Coordinates*
- 关键词：**probability alignment**, **row-adaptive kernels**, **fuzzy union**, **amortized + nonparametric refinement**, **rigidity-consistent**

---

12. 训练/推理伪代码（简要）

```
# 构建老师图（离线）
Q_star = umap_low_kernel(dist(Y_star), a, b)      # 对角置0，并裁剪到(eps,1-eps)

# 训练（逐图）
for batch in loader:
    z, (rho, sigma) = gnn(batch)                  # 视路线而定
    if route_A:
        P_hat = umap_low_kernel(dist(z), a, b)
    else:  # route_B
        q_row = row_kernel(dist(z), rho, sigma)  # exp 或 UMAP 同族
        P_hat = fuzzy_union(q_row)
    loss = CE(Q_star, P_hat, offdiag_mask) + λk*R_smoothk(q_row)
    loss.backward(); optimizer.step()

# 推理
z, (rho, sigma) = gnn(mol)
if route_A:
    P_hat = umap_low_kernel(dist(z), a, b)
    D_hat = ((1/P_hat - 1)/a).clamp_min(0)**(1/(2*b))
    Y0 = MDS_3D(D_hat); Y = CE_refine(P_hat, Y0, a, b)
else:
    P_hat = fuzzy_union(row_kernel(dist(z), rho, sigma))
    Y0 = random_or_spectral_init(mol)             # 任意轻量初始化
    Y  = CE_refine(P_hat, Y0, a, b)
```

---

12. 结论

本方法把分子构象生成从"坐标/能量空间"迁移到**概率图空间**，以 $\mathrm{CE}(Q^{*}, \hat{P})$ 驱动的**概率流形对齐**既稳定又可解释。通过**固定核/行自适应核**两类学生图与 GNN 的深度耦合，以及"**摊销预测 $\hat{P}$ + 非参数 CE 微调 Y**"的推理流程，我们在不牺牲刚体不变性的前提下，获得了与等变扩散、距离几何截然不同且互补的路线。

- **新意**：足够（方法视角与目标空间不同于主流）
- **工作量**：中偏大（要跑强基线、写明理论、工程细节）

附：符号与核

- 低维核：

$$Q_{ij}(Y) = \frac{1}{1 + a\,\|y_i - y_j\|^{2b}}$$

- 指数行核：

$$q_{ij}^{(i)} = \exp\Big(-\frac{\max(0,\, d_{ij} - \rho_i)}{\sigma_i}\Big)$$

- UMAP 同族行核：

$$q_{ij}^{(i)} = \frac{1}{1 + a\left(\max\left(0, \frac{d_{ij} - \rho_i}{\sigma_i}\right)\right)^{2b}}$$

- Fuzzy union：

$$P_{ij} = q_{ij}^{(i)} + q_{ji}^{(j)} - q_{ij}^{(i)}\, q_{ji}^{(j)}$$

- 交叉熵（数值稳定需裁剪 $\hat{p}$ 到 $(\varepsilon, 1-\varepsilon)$）：

$$\mathrm{CE}(p, \hat{p}) = -\,p \log \hat{p} - (1-p)\log(1-\hat{p}).$$

附录 A：梯度推导（CE，对 Y 与 GNN 参数）

A.1 记号与目标

- 老师图 $Q^{\setminus *}$（由真实 $Y^{\setminus *}$ 用 **UMAP 低维核**构造）：

$$Q_{ij}^{\setminus *} = \frac{1}{1 + a\,\|y_i^{\setminus *} - y_j^{\setminus *}\|^{2b}}, \quad Q_{ii}^{\setminus *} = 0.$$

- 学生图 $\hat{P}$：由 GNN 预测（见 A.3 与 A.4）。
- 训练目标（监督）：

$$\mathcal{L}_{\text{train}} = \frac{1}{|S|} \sum_{(i,j)\in S} \Big[-\,Q_{ij}^{\setminus *} \log \hat{P}_{ij} - (1 - Q_{ij}^{\setminus *}) \log(1 - \hat{P}_{ij}) \Big],$$

  其中 $S = \{(i,j) \mid i \neq j\}$（可结合 hop 掩码/负采样）。

- 推理微调（非参数几何解码）：给定 $\hat{P}$，最小化

$$\mathcal{L}_{\text{refine}}(Y) = \mathrm{CE}\big(\hat{P},\, Q(Y)\big), \qquad Q_{ij}(Y) = \frac{1}{1 + a\,\|y_i - y_j\|^{2b}}.$$

A.2 低维核 $Q(Y)$ 对 Y 的梯度（用于推理微调）

令 $d_{ij} = \|y_i - y_j\|$。有

$$Q_{ij} = \frac{1}{1 + a\, d_{ij}^{2b}}, \qquad \frac{\partial Q_{ij}}{\partial d_{ij}} = -\frac{2ab\, d_{ij}^{2b-1}}{\left(1 + a\, d_{ij}^{2b}\right)^2} = -2ab\, d_{ij}^{2b-1}\, Q_{ij}^2.$$

交叉熵对 Q 的导数：

$$\frac{\partial \mathrm{CE}}{\partial Q_{ij}} = -\frac{P_{ij}}{Q_{ij}} + \frac{1 - P_{ij}}{1 - Q_{ij}},$$

这里 P 是目标概率图（推理时 $P=\hat{P}$）。

链式对距离：

$$\frac{\partial \mathrm{CE}}{\partial d_{ij}} = \frac{\partial \mathrm{CE}}{\partial Q_{ij}} \cdot \frac{\partial Q_{ij}}{\partial d_{ij}}.$$

由 $\partial d_{ij}/\partial y_i = (y_i - y_j)/d_{ij}$，得到

$$\frac{\partial \mathrm{CE}}{\partial y_i} = \sum_{j\neq i} \frac{\partial \mathrm{CE}}{\partial d_{ij}} \cdot \frac{y_i - y_j}{d_{ij}}.$$

> 实作要点：对角不参与；对 Q 做 clip $\in (10^{-12}, 1-10^{-12})$；对 d 加 ε 防零除。

A.3 路线 A：固定 UMAP 低维核，从 GNN 嵌入 z 得 $\hat{P}$

令 $r_{ij} = \|z_i - z_j\|$，

$$\hat{P}_{ij} = \frac{1}{1 + a\, r_{ij}^{2b}}, \qquad \frac{\partial \hat{P}_{ij}}{\partial r_{ij}} = -\frac{2ab\, r_{ij}^{2b-1}}{(1 + a\, r_{ij}^{2b})^2}.$$

有

$$\frac{\partial \mathcal{L}_{\text{train}}}{\partial \hat{P}_{ij}} = -\frac{Q_{ij}^{\backslash *}}{\hat{P}_{ij}} + \frac{1 - Q_{ij}^{\backslash *}}{1 - \hat{P}_{ij}}.$$

链式对 r_{ij}：

$$\frac{\partial \mathcal{L}_{\text{train}}}{\partial r_{ij}} = \frac{\partial \mathcal{L}_{\text{train}}}{\partial \hat{P}_{ij}} \cdot \frac{\partial \hat{P}_{ij}}{\partial r_{ij}}.$$

又 $\partial r_{ij}/\partial z_i = (z_i - z_j)/r_{ij}$，于是

$$\frac{\partial \mathcal{L}_{\text{train}}}{\partial z_i} = \sum_{j\neq i} \frac{\partial \mathcal{L}_{\text{train}}}{\partial r_{ij}} \cdot \frac{z_i - z_j}{r_{ij}}.$$

（注意矩阵是对称的，实际实现用张量广播一次性求和会更简洁。）

A.4 路线 B：行自适应核 + Fuzzy Union，从 z, ρ, σ 得 $\hat{P}$

记 $d_{ij} = \|z_i - z_j\|$，$s_{ij} = \max(0,\, d_{ij} - \rho_i)$。

B.1 指数行核

$$q_{ij}^{(i)} = \exp\Big(-\frac{s_{ij}}{\sigma_i}\Big).$$

当 $s_{ij} > 0$ 时：

$$\frac{\partial q_{ij}^{(i)}}{\partial d_{ij}} = -\frac{q_{ij}^{(i)}}{\sigma_i}, \quad \frac{\partial q_{ij}^{(i)}}{\partial \rho_i} = +\frac{q_{ij}^{(i)}}{\sigma_i}, \quad \frac{\partial q_{ij}^{(i)}}{\partial \sigma_i} = q_{ij}^{(i)} \frac{s_{ij}}{\sigma_i^2}.$$

当 $s_{ij} \le 0$ 时，上式为 0（取 ReLU 的 0 子梯度）。

B.2 UMAP 同族行核

$$q_{ij}^{(i)} = \frac{1}{1 + a\Big(\frac{s_{ij}}{\sigma_i}\Big)^{2b}} \quad (s_{ij} > 0 \text{ 时有效})$$

令 $u_{ij} = \frac{s_{ij}}{\sigma_i}$。有

$$\frac{\partial q_{ij}^{(i)}}{\partial u_{ij}} = -\frac{2ab\, u_{ij}^{2b-1}}{(1 + a\, u_{ij}^{2b})^2}.$$

链式：

$$\frac{\partial q}{\partial d_{ij}} = \frac{1}{\sigma_i}\frac{\partial q}{\partial u_{ij}}, \quad \frac{\partial q}{\partial \rho_i} = -\frac{1}{\sigma_i}\frac{\partial q}{\partial u_{ij}}, \quad \frac{\partial q}{\partial \sigma_i} = \frac{\partial q}{\partial u_{ij}} \cdot \frac{\partial u}{\partial \sigma_i} = \frac{\partial q}{\partial u_{ij}} \cdot \Big(-\frac{s_{ij}}{\sigma_i^2}\Big).$$

（当 $s_{ij} \le 0$ 时全为 0。）

B.3 Fuzzy union 对称化

$$\hat{P}_{ij} = q_{ij}^{(i)} + q_{ji}^{(j)} - q_{ij}^{(i)}\, q_{ji}^{(j)}.$$

于是

$$\frac{\partial \hat{P}_{ij}}{\partial q_{ij}^{(i)}} = 1 - q_{ji}^{(j)}, \qquad \frac{\partial \hat{P}_{ij}}{\partial q_{ji}^{(j)}} = 1 - q_{ij}^{(i)}.$$

再乘上 $\partial \mathcal{L}/\partial \hat{P}_{ij}$，即可把梯度**分流**到两条行核支路上；继续链式到 d_{ij}、ρ_i, σ_i、以及 z_i（通过 $\partial d/\partial z$）。

附录 B：从 GNN 到 P 的实现（公式 + 代码）

B.1 公式总览

- **路线 A（固定核）**：

$$z_i = \mathrm{GNN}(G)_i,\;\; r_{ij} = \|z_i - z_j\|,\;\; \hat{P}_{ij} = \frac{1}{1 + a\, r_{ij}^{2b}}.$$

- **路线 B（行自适应核 + union）**：

$$h_i = \mathrm{GNN}(G)_i,\;\; z_i = \mathrm{MLP}_z(h_i),\;\; \rho_i = \mathrm{softplus}(\mathrm{MLP}_\rho(h_i)) + \epsilon,\;\; \sigma_i = \mathrm{softplus}(\mathrm{M}$$

$$d_{ij} = \|z_i - z_j\|,\;\; s_{ij} = \max(0,\, d_{ij} - \rho_i),\;\; q_{ij}^{(i)} = \begin{cases} \exp(-s_{ij}/\sigma_i) & (\exp) \\ \dfrac{1}{1 + a\,(s_{ij}/\sigma_i)^{2b}} & (\mathrm{UMAP}\ \text{同族} \end{cases}$$

$$\hat{P}_{ij} = q_{ij}^{(i)} + q_{ji}^{(j)} - q_{ij}^{(i)}\, q_{ji}^{(j)}.$$

> 训练损失：$\mathcal{L} = \mathrm{CE}(Q^{\backslash *}, \hat{P}) + \lambda_k\, \mathcal{R}_{\text{smooth-k}}(q)$（可选）。

B.2 PyTorch 代码骨架（两条路线可切换）

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GINEConv

EPS = 1e-12

def pairwise_dists(X, eps=EPS):           # X: [n,p]
    diff = X.unsqueeze(1) - X.unsqueeze(0)
    return torch.sqrt((diff*diff).sum(-1) + eps)

def ce_loss_stable(P_star, P_hat, mask_offdiag):
    q = torch.clamp(P_hat, 1e-12, 1-1e-12)
    p = P_star[mask_offdiag]; q = q[mask_offdiag]
    return -(p*torch.log(q) + (1-p)*torch.log(1-q)).mean()

# ---------- 路线 A：固定 UMAP 低维核 ----------
class UMAPLowKernel(nn.Module):
    def __init__(self, a, b):
        super().__init__(); self.a=a; self.b=b
    def forward(self, D):                 # D: [n,n]
        Q = 1.0 / (1.0 + self.a * torch.clamp(D,
min=EPS).pow(2*self.b))
        Q = torch.clamp(Q, 1e-12, 1-1e-12)
        Q.fill_diagonal_(0.0)
        return Q

class QFixedKernelGNN(nn.Module):
    def __init__(self, in_dim, edge_dim, hidden=128, layers=4,
z_dim=32, a=1.6, b=0.8):
        super().__init__()
        self.kernel = UMAPLowKernel(a,b)
        self.xemb = nn.Linear(in_dim, hidden)
        self.eemb = nn.Linear(edge_dim, hidden)
        self.convs = nn.ModuleList([
            GINEConv(nn.Sequential(nn.Linear(hidden,hidden), nn.ReLU(),
                                   nn.Linear(hidden,hidden)))
```

```
            for _ in range(layers)
        ])
        self.proj = nn.Sequential(nn.Linear(hidden,hidden), nn.ReLU(),
                                  nn.Linear(hidden,z_dim))

    def forward(self, x, edge_index, edge_attr):
        h = self.xemb(x); e = self.eemb(edge_attr)
        for conv in self.convs:
            h = F.relu(h + conv(h, edge_index, e))
        z = self.proj(h)                    # [N, zdim]
        D = pairwise_dists(z)               # r_ij
        P_hat = self.kernel(D)              # 1/(1+a r^{2b})
        return P_hat, {"z":z, "D":D}

# ---------- 路线 B：行自适应核 + fuzzy union ----------
class RowAdaptiveHead(nn.Module):
    def __init__(self, family='exp', a=1.6, b=0.8):
        super().__init__(); self.family=family; self.a=a; self.b=b

    def qi_exp(self, D, rho, sigma):
        # q_i(d) = exp( - relu(d - rho_i) / sigma_i )
        X = torch.relu(D - rho[:, None])
        return torch.exp(- X / (sigma[:, None] + EPS))

    def qi_umap(self, D, rho, sigma):
        # q_i(d) = 1 / (1 + a * (relu((d-rho)/sigma))^{2b})
        X = torch.relu(D - rho[:, None]) / (sigma[:, None] + EPS)
        return 1.0 / (1.0 + self.a * torch.clamp(X,
min=0).pow(2*self.b))

    def forward(self, D, rho, sigma, cand_mask=None):
        qi = self.qi_exp(D, rho, sigma) if self.family=='exp' else
self.qi_umap(D, rho, sigma)
        if cand_mask is not None:           # 非候选置零（可选）
            qi = qi * cand_mask.float()
        qj = qi.transpose(0,1)
        P = qi + qj - qi * qj               # fuzzy union
        P = torch.clamp(P, 1e-12, 1-1e-12)
        P.fill_diagonal_(0.0)
        return P, qi

class QRowAdaptiveGNN(nn.Module):
    def __init__(self, in_dim, edge_dim, hidden=128, layers=4,
z_dim=32, family='exp', a=1.6, b=0.8):
        super().__init__()
        self.head = RowAdaptiveHead(family, a, b)
        self.xemb = nn.Linear(in_dim, hidden)
        self.eemb = nn.Linear(edge_dim, hidden)
        self.convs = nn.ModuleList([
            GINEConv(nn.Sequential(nn.Linear(hidden,hidden), nn.ReLU(),
                                   nn.Linear(hidden,hidden)))
            for _ in range(layers)
        ])
        self.proj = nn.Sequential(nn.Linear(hidden,hidden), nn.ReLU(),
                                  nn.Linear(hidden,z_dim))
        self.rho_head    = nn.Sequential(nn.Linear(hidden,hidden),
```

```
nn.ReLU(), nn.Linear(hidden,1))
        self.sigma_head = nn.Sequential(nn.Linear(hidden,hidden),
nn.ReLU(), nn.Linear(hidden,1))
        self.softplus = nn.Softplus()

    def forward(self, x, edge_index, edge_attr, cand_mask=None):
        h = self.xemb(x); e = self.eemb(edge_attr)
        for conv in self.convs:
            h = F.relu(h + conv(h, edge_index, e))
        z = self.proj(h)                                  # [N, zdim]
        D = pairwise_dists(z)                             # [N, N]
        rho   = self.rho_head(h).squeeze(-1)              # [N]
        sigma = self.softplus(self.sigma_head(h).squeeze(-1)) + 1e-3
        P_hat, qi = self.head(D, rho, sigma, cand_mask=cand_mask)
        return P_hat, {"z":z, "D":D, "rho":rho, "sigma":sigma, "qi":qi}
```

训练步骤（逐图求 CE）：

```
def train_one_batch(model, data, optimizer):
    model.train()
    P_hat, aux = model(data.x, data.edge_index, data.edge_attr,
                       cand_mask=getattr(data, "cand_mask", None))
    N = P_hat.size(0)
    mask_off = ~torch.eye(N, dtype=torch.bool, device=P_hat.device)
    loss = ce_loss_stable(data.Q_star, P_hat, mask_off)   # Q_star 为缓
存老师图
    # 可选 smooth-k 正则（行自适应时）
    if "qi" in aux:
        k_target = torch.full((N,), 6.0, device=P_hat.device)
        s = aux["qi"].sum(dim=1)                          # 行强度
        loss = loss + 0.1 * F.mse_loss(s, torch.log2(k_target+1.0))
    optimizer.zero_grad(); loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), 5.0)
    optimizer.step()
    return float(loss.item())
```

B.3 数值稳定建议（避免 NaN）

- 概率裁剪：$\hat{P} \leftarrow \mathrm{clip}(\hat{P}, 10^{-12}, 1-10^{-12})$，对角置 0；
- 距离：$d = \sqrt{\|x\|^2 + \varepsilon}$，$\varepsilon \approx 10^{-12}$；
- 行自适应核：$\sigma = \mathrm{softplus}(\cdot) + 10^{-3}$；当候选邻居数 ≤ 2 的行可**冻结为等权**；
- 训练早期用**较小学习率**，必要时对 $\hat{P}$ 做温度匹配（如对 logits 加温度）；
- 大分子：用 hop 掩码 + 远程负采样（每节点采样 K 个 j）把 $O(N^2)$ 压到 $O(NK)$。

附录 C：从P到Y的微调实现（代码）

1) 推理微调：`refine_Y_from_P`

```
# ========================
#  CE(P_hat, Q(Y)) 的推理微调
```

```
# ========================
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F

EPS = 1e-12

def pairwise_dists_torch(X, eps=EPS):
    """X: [n,3] -> D: [n,n]"""
    diff = X.unsqueeze(1) - X.unsqueeze(0)
    return torch.sqrt((diff * diff).sum(-1) + eps)

class UMAPLowKernel(nn.Module):
    """
    低维核: Q_ij(Y) = 1 / (1 + a * d_ij^{2b})
    """
    def __init__(self, a, b):
        super().__init__()
        self.a = float(a); self.b = float(b)

    def forward(self, D):
        Q = 1.0 / (1.0 + self.a * torch.clamp(D, min=EPS).pow(2.0 *
self.b))
        Q = torch.clamp(Q, 1e-12, 1 - 1e-12)
        Q.fill_diagonal_(0.0)
        return Q

def ce_loss_offdiag(P_tgt, Q, mask_offdiag):
    """
    稳定的 CE: 只在 off-diagonal 上计算
    """
    q = torch.clamp(Q, 1e-12, 1-1e-12)
    p = torch.clamp(P_tgt, 1e-12, 1-1e-12)
    return -(p[mask_offdiag] * torch.log(q[mask_offdiag]) +
             (1 - p[mask_offdiag]) * torch.log(1 -
q[mask_offdiag])).mean()

def center_and_rescale_torch(Y, eps=1e-6):
    with torch.no_grad():
        Y -= Y.mean(dim=0, keepdim=True)
        # 统一一个 RMS 尺度，避免数值漂移
        rms = Y.pow(2).mean().sqrt()
        Y /= (rms + eps)
    return Y

def refine_Y_from_P(P_hat, a, b, steps=50, lr=1e-2,
                    Y0=None, init="random", device="cpu",
                    grad_clip=5.0, verbose=False):
    """
    输入:
      - P_hat: 目标概率图 (numpy 或 torch.float32 [n,n])
      - (a, b): UMAP 低维核参数（和训练保持一致）
      - steps, lr: 微调步数与学习率
      - Y0: 可选初值（[n,3]），若为 None 则根据 init 初始化
      - init: "random" / "spectral"（若你有谱初始化）/ 其它
```

```
    - grad_clip: 梯度裁剪阈值
输出:
    - Y: numpy 数组 [n,3]
    - hist: 每步的 CE 损失列表
"""
# --- 准备数据
if isinstance(P_hat, np.ndarray):
    P_tgt = torch.tensor(P_hat, dtype=torch.float32, device=device)
else:
    P_tgt = P_hat.to(device=device, dtype=torch.float32)

n = P_tgt.size(0)
eye = torch.eye(n, dtype=torch.bool, device=device)
mask_off = ~eye

# --- 初始化 Y
if Y0 is None:
    if init == "random":
        Y = torch.randn(n, 3, device=device) * 0.01
    else:
        # 兜底: 随机
        Y = torch.randn(n, 3, device=device) * 0.01
else:
    if isinstance(Y0, np.ndarray):
        Y = torch.tensor(Y0, dtype=torch.float32, device=device)
    else:
        Y = Y0.to(device=device, dtype=torch.float32)
Y.requires_grad_(True)
center_and_rescale_torch(Y)

kernel = UMAPLowKernel(a, b).to(device)
opt = torch.optim.Adam([Y], lr=lr)
hist = []

for t in range(steps):
    D = pairwise_dists_torch(Y)
    Q = kernel(D)
    loss = ce_loss_offdiag(P_tgt, Q, mask_off)
    opt.zero_grad()
    loss.backward()
    # 梯度裁剪（避免大步震荡）
    if grad_clip is not None:
        torch.nn.utils.clip_grad_norm_([Y], grad_clip)
    opt.step()
    center_and_rescale_torch(Y)
    val = float(loss.item())
    hist.append(val)
    if verbose and (t % 10 == 0 or t == steps - 1):
        print(f"[refine {t:03d}] CE={val:.6f}")

return Y.detach().cpu().numpy(), hist
```

使用示例（单分子）:

```
# 假设你已有 P_hat (numpy [n,n]), 以及 UMAP 低维核参数 a,b
Y_pred, hist = refine_Y_from_P(P_hat, a, b, steps=50, lr=1e-2,
init="random", device="cpu", verbose=True)
```

2) SMACOF-3D 初始化（含解析反解）

场景：当你的学生图来自"固定低维核（路线 A）"，$\hat{P} = 1/(1 + ar^{2b})$ 可**解析反解**出距离矩阵 $\hat{D}$，再用 MDS/SMACOF 得到一个更稳的初值 Y_0，最后喂给 `refine_Y_from_P` 微调十几步即可。

```
# ========================
#   解析反解 + MDS/SMACOF
# ========================
import numpy as np

def invert_umap_q(P, a, b, eps=1e-8):
    """
    从固定 UMAP 低维核的概率 P 反解欧氏距离：
    D_ij = ((1/P_ij - 1)/a)^(1/(2b))
    注意：仅适用于"固定核路线"，行自适应核+fuzzy union不可直接反解。
    """
    P = np.asarray(P, dtype=np.float64)
    P = np.clip(P, 1e-8, 1 - 1e-8)
    X = (1.0 / P - 1.0) / float(a)
    D = np.power(np.clip(X, 0.0, None), 1.0 / (2.0 * float(b)))
    np.fill_diagonal(D, 0.0)
    return D

def classical_mds_3d(D):
    """
    经典 MDS (Torgerson)：Double-centering + 特征分解，取前 3 个正特征。
    D: [n,n] 欧氏距离矩阵
    """
    D2 = D ** 2
    n = D.shape[0]
    J = np.eye(n) - np.ones((n, n)) / n
    B = -0.5 * J @ D2 @ J
    # 特征分解
    w, V = np.linalg.eigh(B)
    idx = np.argsort(w)[::-1]    # 降序
    w = w[idx]
    V = V[:, idx]
    # 取前 3 个非负特征
    pos = w[:3].clip(min=0.0)
    L = np.diag(np.sqrt(pos + 1e-12))
    X = V[:, :3] @ L
    # 中心化（B 已经中心化，理论上均值为 0）
    X -= X.mean(axis=0, keepdims=True)
    # 归一化（稳定后续 CE）
    X /= (np.sqrt((X**2).mean()) + 1e-6)
    return X

def smacof_3d(D, Y0=None, max_iter=200, tol=1e-6, verbose=False,
random_state=0):
    """
    轻量 SMACOF（等权）：最小化应力 sum_{i<j} (d_ij(X) - D_ij)^2
    参考：Borg & Groenen, Modern Multidimensional Scaling
    """
```

```
    rng = np.random.default_rng(random_state)
    n = D.shape[0]
    if Y0 is None:
        X = rng.normal(scale=1e-2, size=(n, 3))
    else:
        X = np.array(Y0, dtype=np.float64)

    D = np.asarray(D, dtype=np.float64)
    W = np.ones_like(D) - np.eye(n)  # 等权重
    eps = 1e-12

    def stress(X):
        diff = X[:, None, :] - X[None, :, :]
        d = np.sqrt((diff**2).sum(-1) + eps)
        return 0.5 * (W * (d - D)**2).sum()

    prev = stress(X)
    for it in range(max_iter):
        # 计算 d(X) 与 B(X)
        diff = X[:, None, :] - X[None, :, :]
        d = np.sqrt((diff**2).sum(-1) + eps)
        R = W * (D / d)
        np.fill_diagonal(R, 0.0)
        B = -R
        B[np.diag_indices(n)] = -B.sum(axis=1)
        # 更新（等权情形下约化成 (1/n) B X）
        X = (1.0 / n) * (B @ X)
        # 居中+归一
        X -= X.mean(axis=0, keepdims=True)
        X /= (np.sqrt((X**2).mean()) + 1e-6)

        cur = stress(X)
        if verbose and (it % 10 == 0 or it == max_iter - 1):
            print(f"[SMACOF {it:03d}] stress={cur:.6f}")
        if abs(prev - cur) < tol * (1.0 + prev):
            break
        prev = cur
    return X
```

route-A 的完整推理范式（$\hat{P} \Rightarrow Y$）：

```
# 1) 从固定核的 P_hat 解析反解距离
D_hat = invert_umap_q(P_hat, a, b)

# 2) 用 MDS/SMACOF 得到一个稳定的初值 Y0
Y0_cmds = classical_mds_3d(D_hat)       # 或用 smacof_3d(D_hat)

# 3) 再做 10~50 步 CE 微调（统一流程）
Y, hist = refine_Y_from_P(P_hat, a, b, steps=30, lr=1e-2, Y0=Y0_cmds,
device="cpu", verbose=True)
```