import torch

import torch.nn as nn

import torch.nn.functional as F

from torch\_geometric.nn import MessagePassing

class LightGCNConv(MessagePassing):

def \_\_init\_\_(self, in\_channels, edge\_feat\_dim, out\_channels, \*\*kwargs):

super(LightGCNConv, self).\_\_init\_\_(aggr='add', \*\*kwargs)

self.edge\_feat\_dim = edge\_feat\_dim

self.lin = nn.Linear(in\_channels + edge\_feat\_dim, out\_channels)

def forward(self, x, edge\_index, edge\_attr):

row, col = edge\_index

out = torch.cat([x[row], edge\_attr.view(-1, self.edge\_feat\_dim), x[col]], dim=-1)

out = self.lin(out)

return self.propagate(edge\_index, x=out)

def message(self, x\_j):

return x\_j

class RecSysGNN(nn.Module):

def \_\_init\_\_(self, feat\_dim, edge\_feat\_dim, latent\_dim, num\_layers, num\_users, num\_items):

super(RecSysGNN, self).\_\_init\_\_()

self.embedding = nn.Embedding(num\_users + num\_items, latent\_dim)

self.convs = nn.ModuleList()

for i in range(num\_layers):

conv = LightGCNConv(feat\_dim, edge\_feat\_dim, latent\_dim)

self.convs.append(conv)

def forward(self, edge\_index, user\_feats, item\_feats, edge\_feats):

user\_emb, item\_emb = self.embedding.weight.split([num\_users, num\_items])

for conv in self.convs:

user\_emb = conv(user\_emb, edge\_index, edge\_feats)

item\_emb = conv(item\_emb, edge\_index, edge\_feats)

return user\_emb, item\_emb

# Synthentic data

num\_users = 100

num\_items = 100

feat\_dim = 64

edge\_feat\_dim = 32

user\_feats = torch.randn(num\_users, feat\_dim)

item\_feats = torch.randn(num\_items, feat\_dim)

# Synthetic edge features, representing user-item interactions

edge\_index = torch.tensor([[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]]) # Example edge index (user-item pairs)

edge\_feats = torch.randn(edge\_index.size(1), edge\_feat\_dim)

# Create the RecSysGNN model

latent\_dim = 64

num\_layers = 3

model = RecSysGNN(feat\_dim, edge\_feat\_dim, latent\_dim, num\_layers, num\_users, num\_items)

# Forward pass

user\_emb, item\_emb = model(edge\_index, user\_feats, item\_feats, edge\_feats)

print("User Embeddings Shape:", user\_emb.shape)

print("Item Embeddings Shape:", item\_emb.shape)