**Code：**

import torch

import torch.nn as nn

import torch.nn.functional as F

from transformers import BertModel, BertTokenizer

class KnowledgeEmbedding(nn.Module):

Knowledge Graph Embedding Layer

def \_\_init\_\_(self, num\_entities, num\_relations, embed\_dim=256):

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

self.entity\_embed = nn.Embedding(num\_entities, embed\_dim)

self.relation\_embed = nn.Embedding(num\_relations, embed\_dim)

nn.init.xavier\_uniform\_(self.entity\_embed.weight)

nn.init.xavier\_uniform\_(self.relation\_embed.weight)

def forward(self, entity\_ids, relation\_ids):

entity\_emb = self.entity\_embed(entity\_ids) # (batch, seq\_len, embed\_dim)

relation\_emb = self.relation\_embed(relation\_ids) # (batch, seq\_len, embed\_dim)

return entity\_emb, relation\_emb

class DynamicAttention(nn.Module):

Context-Aware Knowledge Attention

def \_\_init\_\_(self, embed\_dim=256):

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

self.query = nn.Linear(embed\_dim, embed\_dim)

self.key = nn.Linear(embed\_dim, embed\_dim)

def forward(self, text\_emb, kg\_emb):

# text\_emb: (batch, seq\_len, embed\_dim)

# kg\_emb: (batch, num\_kg\_entities, embed\_dim)

Q = self.query(text\_emb) # (batch, seq\_len, embed\_dim)

K = self.key(kg\_emb) # (batch, num\_kg\_entities, embed\_dim)

attn\_scores = torch.matmul(Q, K.transpose(1, 2)) # (batch, seq\_len, num\_kg\_entities)

attn\_weights = F.softmax(attn\_scores, dim=-1)

# Weighted sum of knowledge entities

context = torch.matmul(attn\_weights, kg\_emb) # (batch, seq\_len, embed\_dim)

return context

class KEGF(nn.Module):

Knowledge-Enhanced Generation Framework

def \_\_init\_\_(self, bert\_model="bert-base-uncased", num\_entities=10000, num\_relations=500):

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

# Text Encoder (BERT-based)

self.bert = BertModel.from\_pretrained(bert\_model)

self.text\_proj = nn.Linear(768, 256) # Project BERT output to 256-dim

# Knowledge Embedding Module

self.kg\_embed = KnowledgeEmbedding(num\_entities, num\_relations, 256)

# Dynamic Attention Layer

self.attention = DynamicAttention(256)

# Decoder (Transformer-based)

self.decoder = nn.TransformerDecoder(

nn.TransformerDecoderLayer(d\_model=256, nhead=8),

num\_layers=3

)

self.vocab\_proj = nn.Linear(256, self.bert.config.vocab\_size)

def forward(self, input\_ids, attention\_mask, entity\_ids, relation\_ids):

# Step 1: Text Encoding

text\_outputs = self.bert(input\_ids, attention\_mask=attention\_mask)

text\_emb = self.text\_proj(text\_outputs.last\_hidden\_state) # (batch, seq\_len, 256)

# Step 2: Knowledge Embedding

entity\_emb, relation\_emb = self.kg\_embed(entity\_ids, relation\_ids)

kg\_emb = entity\_emb + relation\_emb # Simplified fusion

# Step 3: Dynamic Knowledge Attention

context = self.attention(text\_emb, kg\_emb)

# Step 4: Decoding with Knowledge Context

memory = torch.cat([text\_emb, context], dim=2) # Enhanced representation

tgt = torch.zeros\_like(text\_emb) # Placeholder for autoregressive decoding

decoder\_output = self.decoder(tgt, memory)

# Step 5: Vocabulary Projection

logits = self.vocab\_proj(decoder\_output) # (batch, seq\_len, vocab\_size)

return logits

# Example Usage

if \_\_name\_\_ == "\_\_main\_\_":

tokenizer = BertTokenizer.from\_pretrained("bert-base-uncased")

model = KEGF(num\_entities=10000, num\_relations=500)

# Sample Input

text = "Einstein proposed the theory of relativity."

inputs = tokenizer(text, return\_tensors="pt", padding=True, truncation=True)

entity\_ids = torch.randint(0, 10000, (1, 10)) # Mock entity IDs (batch\_size=1, seq\_len=10)

relation\_ids = torch.randint(0, 500, (1, 10)) # Mock relation IDs

# Forward Pass

logits = model(

input\_ids=inputs["input\_ids"],

attention\_mask=inputs["attention\_mask"],

entity\_ids=entity\_ids,

relation\_ids=relation\_ids

)

print("Output logits shape:", logits.shape)

**Code interpretation**

**1.KnowledgeEmbedding Class**

**Purpose**: Embeds entities and relations from the knowledge graph into dense vectors.

**Key Components**:

entity\_embed: Embedding layer for entities.

relation\_embed: Embedding layer for relations.

**Forward Pass**: Takes entity and relation IDs, returns their embeddings.

**2.DynamicAttention Class**

**Purpose**: Computes attention weights between text tokens and knowledge entities.

**Key Components**:

query and key: Linear layers to project text and knowledge embeddings.

**Forward Pass**: Computes attention scores using dot-product attention and returns a context vector.

**3.KEGF Class (Main Model)**

**Purpose**: Integrates text encoding, knowledge fusion, and generation.

**Components**:

**Text Encoder**: Uses a pre-trained BERT model to encode input text.

**Knowledge Embedding**: Maps KG entities/relations to embeddings.

**Dynamic Attention**: Fuses text and knowledge embeddings contextually.

**Decoder**: Transformer-based decoder for text generation.

**Workflow**:

Encode text with BERT and project to lower dimension.

Embed KG entities and relations.

Compute attention between text and KG embeddings.

Decode the fused representation to generate text.

**4.Example Usage**

**Input Preparation**: Tokenizes text and generates mock entity/relation IDs.

**Forward Pass**: Demonstrates end-to-end model execution.

**Output**: Logits for vocabulary prediction (shape [batch\_size, seq\_len, vocab\_size]).

**Key Features**

**Modular Design**: Separates text encoding, knowledge embedding, and decoding for clarity.

**Dynamic Knowledge Fusion**: Uses attention to dynamically select relevant knowledge.

**Scalability**: Supports large knowledge graphs via configurable entity/relation counts.

**Compatibility**: Integrates with HuggingFace's BERT for text encoding.

**Usage Notes**

**Data Preparation**: Ensure entity/relation IDs align with your knowledge graph.

**Training**: Use cross-entropy loss with teacher forcing for sequence generation.

**Inference**: Implement beam search or sampling for text generation.

This code provides a foundational implementation of the proposed framework, which can be extended for specific applications like dialogue systems or technical documentation generation.