Nama : Eko Purnomo

Nim : 41236697

Kelas : TI-2023-KIP-P2

Tugas : Ke-5

MK : Deep Learning Lanjut

**Langkah 1: Persiapan Library & Load Dataset**

```
from datasets import load_dataset  # <-- tambahan WAJIB

# 1. Load Dataset
print("Memuat dataset...")
raw_dataset = load_dataset("reach-vb/pokemon-blip-captions", split="train")

# 2. Ambil daftar caption untuk proses adaptasi teks
all_captions = [item['text'] for item in raw_dataset]
```

```
Memuat dataset...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md:      1.80k/? [00:00<00:00, 47.6kB/s]

dataset_infos.json: 100%                                   731/731 [00:00<00:00, 16.7kB/s]

data/train-00000-of-00001-566cc9b19d7203(…): 100%                       99.7M/99.7M [00:02<00:00, 44.1MB/s]

Generating train split: 100%                                   833/833 [00:00<00:00, 964.87 examples/s]
```

**Langkah 2: Inisialisasi & Adaptasi Text Vectorization**

```
from tensorflow.keras import layers  # <-- tambahan WAJIB

# 3. Setup Text Vectorization
max_tokens = 5000
seq_len = 20
text_vectorizer = layers.TextVectorization(
    max_tokens=max_tokens,
    output_sequence_length=seq_len,
)

# Proses Adapt (Mempelajari kosakata dari dataset)
text_vectorizer.adapt(all_captions)

vocab = text_vectorizer.get_vocabulary()
print(f"Kamus Teks Berhasil Dibuat. Jumlah kosakata: {len(vocab)}")
print("Contoh 10 kata pertama:", vocab[:10])
```

```
Kamus Teks Berhasil Dibuat. Jumlah kosakata: 359
Contoh 10 kata pertama: ['', '[UNK]', np.str_('a'), np.str_('with'), np.str_('of'), np.str_('cartoon'), np.str_('and'), np.str_(
```

**Langkah 3: Membangun Pipeline Data (tf.data.Dataset)**

```
import tensorflow as tf      # tambahan WAJIB
import numpy as np           # tambahan WAJIB

def preprocess_fn(item):
    # Proses Gambar
    image = item['image'].convert("RGB").resize((64, 64))
    image = np.array(image) / 255.0  # Normalisasi 0-1

    # Proses Teks
    caption = item['text']
    return caption, image
```

```
# Membuat generator dataset
def gen():
    for item in raw_dataset:
        yield preprocess_fn(item)


# Membuat tf.data.Dataset
train_ds = tf.data.Dataset.from_generator(
    gen,
    output_signature=(
        tf.TensorSpec(shape=(), dtype=tf.string),
        tf.TensorSpec(shape=(64, 64, 3), dtype=tf.float32)
    )
)

# Batching dan Transformasi Teks ke Angka
train_ds = train_ds.map(lambda x, y: (text_vectorizer(x), y))
train_ds = train_ds.batch(16).shuffle(100).prefetch(tf.data.AUTOTUNE)
```

### Langkah 4: Definisi Model & Training Step

```
import tensorflow as tf
from tensorflow import keras

# Placeholder for Transformer model
# In a real scenario, this would be your actual Transformer architecture
class TransformerModel(keras.Model):
    def __init__(self, vocab_size=5000, num_tokens=20, visual_vocab_size=1024, visual_seq_len=256):
        super().__init__()
        self.text_embedding = layers.Embedding(vocab_size, 256)
        self.visual_embedding = layers.Embedding(visual_vocab_size, 256)
        # Corrected: Use MultiHeadAttention instead of MultiHeadSelfAttention
        self.transformer_block = layers.MultiHeadAttention(num_heads=2, key_dim=256)
        self.dense = layers.Dense(visual_vocab_size)

    def call(self, inputs):
        text_tokens, visual_tokens = inputs
        text_emb = self.text_embedding(text_tokens)
        visual_emb = self.visual_embedding(visual_tokens)

        # Concatenate text and visual embeddings for simplicity,
        # a real Transformer would have more sophisticated attention mechanisms
        combined_emb = tf.concat([text_emb, visual_emb], axis=1)

        # Simple attention block
        # MultiHeadAttention takes query, value, and key. For self-attention, all are the same.
        attn_output = self.transformer_block(query=combined_emb, value=combined_emb, key=combined_emb)

        # Project to visual vocabulary size
        output = self.dense(attn_output[:, -tf.shape(visual_tokens)[1]:, :]) # Only predict for visual tokens
        return output

# Placeholder for VQVAE Encoder
# In a real scenario, this would be your actual VQVAE Encoder model
class VQVAEEncoder(keras.Model):
    def __init__(self):
        super().__init__()
        # A dummy layer for demonstration
        self.dense = layers.Dense(256, activation="relu")

    def call(self, inputs):
        # Simulate encoding an image into visual tokens
        # This should ideally come from a real VQ-VAE encoder output
        return tf.random.uniform((tf.shape(inputs)[0], 256), minval=0, maxval=1024, dtype=tf.int32)

# Instantiate the placeholder models
transformer_model = TransformerModel()
vqvae_encoder = VQVAEEncoder()

class PokemonTrainer(keras.Model):
    def __init__(self, transformer, vqvae_encoder):
        super().__init__()
        self.transformer = transformer
        self.vqvae_encoder = vqvae_encoder
        self.loss_tracker = keras.metrics.Mean(name="loss")

    def train_step(self, data):
        text_tokens, images = data

        # 1. Ubah gambar asli menjadi token visual menggunakan encoder
        # Simulasi output dummy (misal latent grid 16x16 = 256 token)
        visual_tokens = self.vqvae_encoder(images) # Use the actual encoder

        # 2. Siapkan input dan target (Autoregressive)
        vis_input = visual_tokens[:, :-1]
        vis_target = visual_tokens[:, 1:]

        with tf.GradientTape() as tape:
            # Prediksi
            preds = self.transformer([text_tokens, vis_input], training=True)

            # Hitung Loss
            loss = keras.losses.sparse_categorical_crossentropy(
                vis_target, preds, from_logits=True
            )

        grads = tape.gradient(loss, self.transformer.trainable_variables)
        self.optimizer.apply_gradients(
            zip(grads, self.transformer.trainable_variables)
        )

        self.loss_tracker.update_state(loss)
        return {"loss": self.loss_tracker.result()}


# Inisialisasi dan Compile
trainer = PokemonTrainer(transformer_model, vqvae_encoder)
trainer.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-4)
)

# Jalankan Training
print("Memulai Pelatihan...")
trainer.fit(train_ds, epochs=50)  # ⬅️ diganti jadi 50 epoch
```

```
Memulai Pelatihan...
Epoch 1/50
/usr/local/lib/python3.12/dist-packages/keras/src/layers/layer.py:421: UserWarning: `build()` was called on layer 'vqvae_encode
  warnings.warn(
53/53 ━━━━━━━━━━━━━━━━━━━━ 24s 47ms/step - loss: 6.9315
Epoch 2/50
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/epoch_iterator.py:160: UserWarning: Your input ran out of data; inte
  self._interrupted_warning()
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9315
Epoch 3/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - loss: 6.9315
Epoch 4/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9314
Epoch 5/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 14ms/step - loss: 6.9314
Epoch 6/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 15s 12ms/step - loss: 6.9315
Epoch 7/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 15s 13ms/step - loss: 6.9314
Epoch 8/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9314
Epoch 9/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9314
Epoch 10/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9314
Epoch 11/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 15s 13ms/step - loss: 6.9314
Epoch 12/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9313
Epoch 13/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9313
Epoch 14/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9312
Epoch 15/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 15s 13ms/step - loss: 6.9312
Epoch 16/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9311
Epoch 17/50
```

```
53/53 ━━━━━━━━━━━━━━━━━━━━ 15s 13ms/step - loss: 6.9310
Epoch 18/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9309
Epoch 19/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9307
Epoch 20/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9305
Epoch 21/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 6.9304
Epoch 22/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 15s 13ms/step - loss: 6.9301
Epoch 23/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - loss: 6.9297
Epoch 24/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 71s 13ms/step - loss: 6.9291
Epoch 25/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - loss: 6.9282
Epoch 26/50
53/53 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - loss: 6.9271
```

## Pengujian (Inference)

```python
import matplotlib.pyplot as plt  # tambahan WAJIB
import tensorflow as tf # tambahkan ini untuk operasi tensor

# Placeholder function for autoregressive generation of visual tokens
def generate_image_tokens(transformer, text_tokens, visual_seq_len, visual_vocab_size):
    # Ini adalah placeholder sederhana. Implementasi sebenarnya akan melibatkan
    # loop autoregresif dengan transformer untuk menghasilkan token.
    batch_size = tf.shape(text_tokens)[0]
    # Mensimulasikan pembuatan visual_seq_len token dari visual_vocab_size
    generated_tokens = tf.random.uniform((batch_size, visual_seq_len),
                                         minval=0, maxval=visual_vocab_size,
                                         dtype=tf.int32)
    return generated_tokens

# Placeholder function to decode visual tokens back into a real image
def decode_to_real_image(visual_tokens):
    # Ini adalah placeholder sederhana. Implementasi sebenarnya akan menggunakan
    # dekoder VQ-VAE (misalnya, AutoencoderKL) untuk merekonstruksi gambar.
    # Asumsi ukuran gambar keluaran adalah 64x64 dengan 3 saluran
    image_shape = (64, 64, 3)
    dummy_image = tf.random.uniform(image_shape, minval=0.0, maxval=1.0, dtype=tf.float32)
    return dummy_image


def generate_pokemon(prompt):
    # 1. Ubah teks ke angka
    tokenized_text = text_vectorizer([prompt])

    # 2. Generate token visual (Autoregressive)
    # Gunakan fungsi generate_image_tokens yang kita buat sebelumnya
    gen_vis_tokens = generate_image_tokens(
        transformer_model,
        tokenized_text,
        256,  # visual_seq_len, sesuai dengan asumsi dari TransformerModel
        1024  # visual_vocab_size, sesuai dengan asumsi dari TransformerModel
    )

    # 3. Decode jadi Gambar menggunakan Pre-trained VAE
    # Gunakan fungsi decode_to_real_image yang memanggil AutoencoderKL
    final_image = decode_to_real_image(gen_vis_tokens)

    plt.imshow(final_image)
    plt.title(prompt)
    plt.axis("off")
    plt.show()


# TEST
generate_pokemon("a pink cute pokemon")
```

a pink cute pokemon