Reg no: 212222090012

Exp 10:Content Creation (Reports, Articles, Case Studies, etc.) Using Prompt Patterns

Aim:

To demonstrate how various prompting techniques (query decomposition, decision-making, semantic filtering, etc.) can be employed to create content such as reports, articles, case studies, or creative works like comic books, using ChatGPT or similar models. The objective is to highlight how different prompt structures affect the content's quality, coherence, and structure.

PROCEDURE:

1. Select a Topic: Choose a topic (e.g., market trends, climate change, sci-fi story) that interests you.
2. Use Basic Prompts: Start by crafting simple, general prompts to generate initial content.
3. Refine Your Prompts: Gradually introduce more complexity by adding details, refining the structure, and using different prompt techniques like decision making or semantic filtering.
4. Evaluate the Output: Review the generated content for clarity, creativity, and accuracy.
5. Iterate for Improvement: Based on the feedback, refine the prompts and regenerate the content as needed.

TOOLS USED:

CHATGPT, GEMINI AI

OUTPUT:

USING AI TOOL FOR MINI PROJECT ON " MEDICAL WASTE DETECTION USING DEEP LEARNING"

1. **First Draft: A basic draft of the report, case study, article, or story generated using simple prompts.**

Prompt: "Create a model for detecting medical waste vs non medical waste"

 Dataset Preparation:

- Organize images into two folders: medical/ and non_medical/.
- Resize images to 50x50 pixels.

 Model Setup:

- Use **MobileNetV2** (transfer learning) as the base model.

- Add a few dense layers with softmax activation for 2-class output.
  Training:
- Use ImageDataGenerator for loading and augmenting images.
- Train for ~10 epochs using adam optimizer and categorical crossentropy loss.
  Prediction:
- Load an image, preprocess it, and run model.predict() to get classification.
- Output: "Medical Waste" or "Non-Medical Waste".
  Optional:
- Save the model using .save().
- Deploy using Flask or Streamlit for a simple interface.

2.**Refined Content: A more detailed and structured output, achieved by applying more advanced prompt techniques.**

Prompt:

“Give a python code for the model”

Medical Waste Detection Model using MobileNetV2

✅ **Assumptions**:

- Your dataset is structured as:

```
CopyEdit
dataset/
├── medical/
│   ├── image1.jpg
│   └── ...
└── non_medical/
    ├── image1.jpg
    └── ...
```

✅ Python Code (model_training.py)

```
python
CopyEdit
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
import numpy as np
import os

# Step 1: Load Data
data_dir = "dataset"
image_size = (50, 50)
batch_size = 32
```

```
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

train_data = datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

val_data = datagen.flow_from_directory(
    data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)

# Step 2: Build Model using Transfer Learning
base_model = MobileNetV2(input_shape=(50, 50, 3), include_top=False,
weights='imagenet')
base_model.trainable = False  # Freeze base layers

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(2, activation='softmax')  # Two classes: medical, non_medical
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Step 3: Train the Model
model.fit(train_data, validation_data=val_data, epochs=10)

# Step 4: Save the Model
model.save("medical_waste_classifier.h5")
```

🖼️ Prediction on a New Image

```
python
CopyEdit
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np

# Load trained model
model = load_model("medical_waste_classifier.h5")

# Load and preprocess the image
img = image.load_img("test.jpg", target_size=(50, 50))
x = image.img_to_array(img) / 255.0
x = np.expand_dims(x, axis=0)

# Make prediction
prediction = model.predict(x)
```

```
class_names = ['medical', 'non_medical']
predicted_class = class_names[np.argmax(prediction)]
print(f"Prediction: {predicted_class} (Confidence: {np.max(prediction) *
100:.2f}%)")
```

3. Multiple Versions: Different versions of the content, showing the effect of prompt changes on the output.

Prompt: " Use deeplearning model"

Medical vs Non-Medical Waste Classifier (Deep Learning CNN from Scratch)

✅ Full Python Code:

```
python
CopyEdit
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

# Step 1: Load and Preprocess Data
img_size = (50, 50)
batch_size = 32

datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

train_data = datagen.flow_from_directory(
    'dataset',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

val_data = datagen.flow_from_directory(
    'dataset',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)

# Step 2: Build Deep Learning CNN Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(50, 50, 3)),
    MaxPooling2D(2, 2),
```

```
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='softmax')  # 2 classes: medical, non_medical
])

# Step 3: Compile the Model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Step 4: Train the Model
model.fit(train_data, validation_data=val_data, epochs=10)

# Step 5: Save the Model
model.save('medical_waste_cnn.h5')
```

🖼️ Test a New Image

```
python
CopyEdit
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

# Load model
model = load_model('medical_waste_cnn.h5')

# Load and preprocess image
img = image.load_img('test.jpg', target_size=(50, 50))
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)

# Predict
class_names = ['medical', 'non_medical']
prediction = model.predict(img_array)
predicted_class = class_names[np.argmax(prediction)]
print(f"Predicted class: {predicted_class} (Confidence: {np.max(prediction)
* 100:.2f}%)")
```

✅ Summary:

- Uses **Convolutional Neural Networks (CNNs)** — a classic deep learning architecture for image classification.
- No transfer learning — model is trained **from scratch**.
- Works with any dataset structured in folders.

Conclusion:

By applying various prompting techniques, you can generate high-quality content for a wide range of use cases, from business reports and case studies to creative works like short stories and articles. This experiment demonstrates how structured prompting can guide AI models like ChatGPT to create coherent, accurate, and engaging outputs tailored to specific needs.

RESULT:

The experiment successfully showed that using structured prompting techniques significantly improves the quality and relevance of AI-generated content. Whether applied to professional tasks such as business reports or creative projects like short stories, well-designed prompts helped guide the AI in producing coherent, context-aware, and purpose-driven outputs. The assistant responded effectively to different formats and objectives, confirming that prompt engineering is a key skill for maximizing the utility of language models like ChatGPT in diverse real-world applications.