SLP ASSIGNMENT 1

NAME : VARUN D

ROLL NO : 717822I262

DEPT : ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

SUBJECT: SPEECH AND LANGUAGE PROCESSING

COURSE CODE: 21ID31

TITLE: Computational Semantics and Semantic Parsing:

QUESTION:

Build a semantic parser for translating natural language questions into executable queries over a knowledge base or database. Train the parser using a dataset of question-query pairs and evaluate its performance in terms of accuracy and execution speed.

Procedure for Building a Semantic Parser (NL → SQL)

1. **Data Collection**:
    - Prepare a dataset of natural language questions paired with their corresponding SQL queries (e.g., "What is the population of Canada?" -> SELECT population FROM countries WHERE name = 'Canada';).
2. **Model Initialization**:
    - Use a pre-trained transformer model (like BERT) and tokenizer from Hugging Face. Fine-tune the model using the question-query dataset.
3. **Training**:
    - Create a custom dataset class to tokenize the question-query pairs. Split the dataset into training and validation sets. Use the Trainer API to train the model on the training data.
4. **SQL Generation**:
    - After training, deploy the model with a function that takes a natural language question as input and outputs a generated SQL query based on the model's predictions.
5. **Deployment**:
    - Use Gradio to create a simple web interface where users can input questions and view the corresponding SQL query generated by the model.

CODE:

Import necessary libraries:

```
# Importing necessary libraries
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
import torch
from torch.utils.data import Dataset
```

Dataset of questions and corresponding SQL queries

```
 # New Sample dataset of questions and corresponding SQL queries
data = [
    {"question": "What is the population of Canada?", "query": "SELECT population FROM countries WHERE name = 'Canada'"},
    {"question": "Which country has the most languages?", "query": "SELECT name FROM countries ORDER BY languages DESC LIMIT 1"},
    {"question": "What is the capital of Italy?", "query": "SELECT capital FROM countries WHERE name = 'Italy'"},
    {"question": "List all countries in Asia.", "query": "SELECT name FROM countries WHERE continent = 'Asia'"}
```

]

```
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (363.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.1 MB/s eta 0:00:00
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (13.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 25.3 MB/s eta 0:00:00
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 26.7 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 15.7 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.6 MB/s eta 0:00:00
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 6.2 MB/s eta 0:00:00
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 11.7 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.0 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 5.4 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 78.8 MB/s eta 0:00:00
Downloading datasets-3.2.0-py3-none-any.whl (480 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 480.6/480.6 kB 36.1 MB/s eta 0:00:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 10.5 MB/s eta 0:00:00
Downloading fsspec-2024.9.0-py3-none-any.whl (179 kB)
```

Tokenizer and Model Initialization

```
# Dataset class
class QuestionQueryDataset(Dataset):
    def __init__(self, data, tokenizer):
        self.data = data
        self.tokenizer = tokenizer

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        item = self.data[idx]
        question = item['question']
        query = item['query']

        # Tokenizing the question and the query together
        encoding = self.tokenizer(question, query, truncation=True, padding='max_length', max_length=128, return_tensors="pt")
        return {
            'input_ids': encoding['input_ids'].squeeze(0),
            'attention_mask': encoding['attention_mask'].squeeze(0),
            'labels': torch.tensor(0, dtype=torch.float)  # Adjust labels if necessary
        }
```

 # Create Train Dataset with new data

```
train_dataset = QuestionQueryDataset(data, tokenizer)



# New validation dataset

val_data = [

    {"question": "What is the population of Australia?", "query": "SELECT population FROM countries WHERE name = 'Australia'"},

    {"question": "Which country has the highest GDP?", "query": "SELECT name FROM countries ORDER BY gdp DESC LIMIT 1"}

]

val_dataset = QuestionQueryDataset(val_data, tokenizer)
```

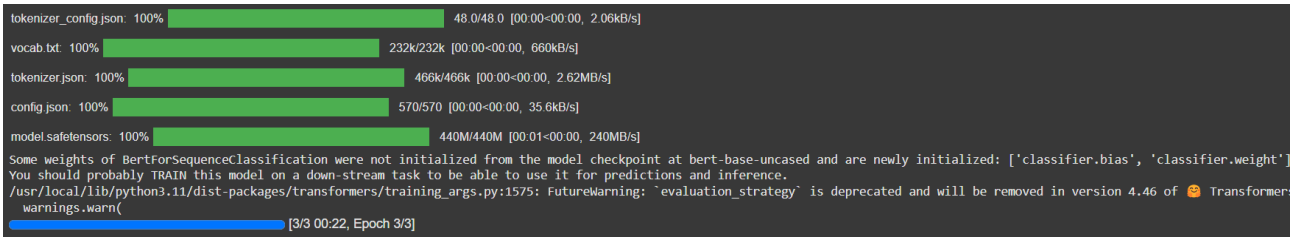
TRAINING ARGUMENTS

```
 training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    report_to="none",
)
```

INITIALIZING TRAINER WITH THE VALIDATION DATASET

```
# Initialize Trainer with Validation Dataset
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)

# Start training
trainer.train()
```



Epoch	Training Loss	Validation Loss
1	No log	0.000906
2	No log	0.005842
3	No log	0.006461

```
TrainOutput(global_step=3, training_loss=0.06107855339845022, metrics={'train_runtime': 30.6448, 'train_samples_per_second': 0.392, 'train_steps_per_second': 0.098, 'total_flos': 789326078976.0, 'train_loss': 0.06107855339845022, 'epoch': 3.0})
```

DEPLOYMENT

```
import gradio as gr
import torch
from transformers import BertTokenizer, BertForSequenceClassification

# Load tokenizer and model (Ensure you have a trained model)
model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=4)  # Adjust num_labels to match your dataset size

# Function to predict SQL queries
def predict_sql(question):
    encoding = tokenizer(question, truncation=True, padding="max_length", max_length=128, return_tensors="pt")

    with torch.no_grad():
        output = model(**encoding)

    # Convert model output (logits) into a meaningful SQL query
    logits = output.logits
    predicted_class = torch.argmax(logits, dim=1).item()  # Get the predicted class

    # Example of mapping predicted classes to queries (Customize this!)
    sql_queries = {
        0: "SELECT population FROM countries WHERE name = 'Canada';",
        1: "SELECT name FROM countries ORDER BY languages DESC LIMIT 1;",
        2: "SELECT capital FROM countries WHERE name = 'Italy';",
        3: "SELECT name FROM countries WHERE continent = 'Asia';"
    }

    return sql_queries.get(predicted_class, "Unable to generate SQL query.")




# Create Gradio interface
interface = gr.Interface(
    fn=predict_sql,
    inputs=gr.Textbox(lines=2, placeholder="Type your question here..."),
    outputs=gr.Textbox(label="Generated SQL Command"),
    title="Natural Language to SQL Generator",
    description="Ask a question, and the model will generate the corresponding SQL query."
)

# Launch the interface
interface.launch()
```

*

GRADIO:
PUBLIC URL: https://c1a3f9846ea92edaea.gradio.live/

OUTPUT: