```
!pip install transformers datasets torch sqlite3
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.42.4)
Collecting datasets
  Downloading datasets-2.20.0-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.3.1+cu121)
ERROR: Could not find a version that satisfies the requirement sqlite3 (from versions: none)
ERROR: No matching distribution found for sqlite3
```

```
!pip install datasets
```

```
Collecting datasets
  Using cached datasets-2.20.0-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets) (3.15.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.26.4)
Collecting pyarrow>=15.0.0 (from datasets)
  Downloading pyarrow-17.0.0-cp310-cp310-manylinux_2_28_x86_64.whl.metadata (3.3 kB)
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/dist-packages (from datasets) (0.6)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (2.1.4)
Collecting requests>=2.32.2 (from datasets)
  Downloading requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.4)
Collecting xxhash (from datasets)
  Downloading xxhash-3.4.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl.metadata (7.2 kB)
Collecting fsspec<=2024.5.0,>=2023.1.0 (from fsspec[http]<=2024.5.0,>=2023.1.0->datasets)
  Downloading fsspec-2024.5.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.10.0)
Requirement already satisfied: huggingface-hub>=0.21.2 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.23.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (2.3.4)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.21.2->d
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (2.0.7
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (2024.
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->datasets) (1
Downloading datasets-2.20.0-py3-none-any.whl (547 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 547.8/547.8 kB 15.2 MB/s eta 0:00:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 12.4 MB/s eta 0:00:00
Downloading fsspec-2024.5.0-py3-none-any.whl (316 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.1/316.1 kB 29.1 MB/s eta 0:00:00
Downloading pyarrow-17.0.0-cp310-cp310-manylinux_2_28_x86_64.whl (39.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 39.9/39.9 MB 19.3 MB/s eta 0:00:00
Downloading requests-2.32.3-py3-none-any.whl (64 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 64.9/64.9 kB 6.8 MB/s eta 0:00:00
Downloading multiprocess-0.70.16-py310-none-any.whl (134 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 13.7 MB/s eta 0:00:00
Downloading xxhash-3.4.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.1/194.1 kB 19.6 MB/s eta 0:00:00
Installing collected packages: xxhash, requests, pyarrow, fsspec, dill, multiprocess, datasets
  Attempting uninstall: requests
    Found existing installation: requests 2.31.0
    Uninstalling requests-2.31.0:
```

```
from datasets import load_dataset

# Load the Predefined dataset
dataset = load_dataset("wikisql")

# Splitting the dataset into train, validation, and test sets
train_dataset = dataset['train']
validation_dataset = dataset['validation']
test_dataset = dataset['test']
```

```
print(dataset)
```

```
DatasetDict({
    test: Dataset({
        features: ['phase', 'question', 'table', 'sql'],
        num_rows: 15878
    })
    validation: Dataset({
        features: ['phase', 'question', 'table', 'sql'],
        num_rows: 8421
    })
    train: Dataset({
        features: ['phase', 'question', 'table', 'sql'],
        num_rows: 56355
    })
})
```

```
import torch
from transformers import T5ForConditionalGeneration, T5Tokenizer, Trainer, TrainingArguments

# Initialize the model and tokenizer
model_name = "t5-small"
tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

```
def preprocess_function(examples):
    inputs = ["translate natural language to SQL: " + query for query in examples['question']]
    targets = [example['human_readable'] for example in examples['sql']]
    model_inputs = tokenizer(inputs, max_length=512, truncation=True, padding="max_length")
    labels = tokenizer(targets, max_length=512, truncation=True, padding="max_length").input_ids
    model_inputs["labels"] = labels
    return model_inputs

# Prepare training data
train_data = train_dataset.map(preprocess_function, batched=True, remove_columns=['phase', 'sql', 'table'])
validation_data = validation_dataset.map(preprocess_function, batched=True, remove_columns=['phase', 'sql', 'table'])
```

```
# Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=1,
    weight_decay=0.01,
)

# Create a Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=validation_data,
)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1494: FutureWarning: `evaluation_strategy` is deprecated and will
  warnings.warn(
```

```
# Fine-tune the model
trainer.train()
```

```
results = trainer.evaluate()
```

```
print(results)
```

```
{'eval_loss': 0.02726702019572258, 'eval_runtime': 253.2536, 'eval_samples_per_second': 33.251, 'eval_steps_per_second': 4.158, 'epoch':
```

```
import sqlite3

# Connect to the database
conn = sqlite3.connect('sample.db')
c = conn.cursor()

# Create a sample table
c.execute('''
CREATE TABLE IF NOT EXISTS employees (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    position TEXT NOT NULL,
    salary INTEGER NOT NULL
)
''')

# Insert sample data
c.execute("INSERT INTO employees (name, position, salary) VALUES ('Alice', 'Manager', 80000)")
c.execute("INSERT INTO employees (name, position, salary) VALUES ('Bob', 'Engineer', 70000)")
c.execute("INSERT INTO employees (name, position, salary) VALUES ('Charlie', 'HR', 60000)")

# Commit changes and close the connection
conn.commit()
conn.close()
```

```
import re
def translate_and_execute(natural_language_query):
    # Tokenize input with padding and truncation
    inputs = tokenizer("translate natural language to SQL: " + natural_language_query, return_tensors="pt", max_length=512, truncation=True,

    # Ensure the attention mask is set
    attention_mask = inputs.get("attention_mask")
    if attention_mask is None:
        attention_mask = torch.ones_like(inputs["input_ids"])

    # Move tensors to the same device as the model
    inputs = {key: val.to(device) for key, val in inputs.items()}
    attention_mask = attention_mask.to(device)  # Ensure attention_mask is on the same device

    model.eval()
    with torch.no_grad():
        outputs = model.generate(
            inputs["input_ids"],
            attention_mask=attention_mask,
            max_new_tokens=100,  # Adjust as needed
            do_sample=False
        )

    sql_query = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Replace "table" with the actual table name "employee"
    sql_query = sql_query.replace("table", "employees")

    # Add single quotes around string literals
    sql_query = re.sub(r"= ([^'\s]+)", r"= '\1'", sql_query)
    sql_query = re.sub(r"= '(\d+)'", r"= \1", sql_query)  # Correct numeric values without quotes

    print("Generated SQL Query:", sql_query)  # Debugging: print the SQL query

    try:
        result = execute_sql_query(sql_query)
    except sqlite3.OperationalError as e:
        print(f"SQL error: {e}")
        result = []

    return result, sql_query

def execute_sql_query(query):
    conn = sqlite3.connect('sample.db')
    c = conn.cursor()
    try:
        c.execute(query)
        result = c.fetchall()
    except sqlite3.OperationalError as e:
        print(f"Error executing query: {e}")
        result = []
    conn.close()
    return result

def format_results_to_natural_language(results):
    if not results:
        return "No results found."
    return "The query returned the following results:\n" + "\n".join([str(result) for result in results])
```

```
# Define the natural language query
natural_language_query = "Show names of the employee where a salary greater than 65000."

# Translate and execute the query
result, sql_query = translate_and_execute(natural_language_query)
print("SQL Query:", sql_query)
print("Result:", result)

# Format the results into natural language
formatted_result = format_results_to_natural_language(result)
print("Formatted Result:", formatted_result)
```

```
Generated SQL Query: SELECT Name FROM employees WHERE Lohn > 65000
Error executing query: no such column: Lohn
SQL Query: SELECT Name FROM employees WHERE Lohn > 65000
Result: []
Formatted Result: No results found.
```

Start coding or generate with AI.