Objective: Convert natural language queries into SQL using an appropriate pre-trained model.

Suggested Models: tscholak/optimum-**nl2sql**

```
import sqlite3
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

# Load model
tokenizer = AutoTokenizer.from_pretrained("tscholak/cxmefzzi")
model = AutoModelForSeq2SeqLM.from_pretrained("tscholak/cxmefzzi")

# convert NL to SQL
def nl_to_sql(nl_query, db_id, schema):
    input_text = f"{nl_query} | {db_id} | {schema}"
    inputs = tokenizer(input_text, return_tensors="pt")
    outputs = model.generate(**inputs, max_length=128)
    sql = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return sql

# Clean model output
def clean_predicted_sql(predicted_sql):
    if "|" in predicted_sql:
        parts = predicted_sql.split("|")
        if len(parts) > 1:
            return parts[1].strip()
    return predicted_sql.strip()


def create_sample_db():
    conn = sqlite3.connect(":memory:")
    cursor = conn.cursor()
    cursor.execute("CREATE TABLE customers (id INTEGER, name TEXT, city TEXT, age INTEGER);")
    sample_data = [
        (1, 'Aditya', 'New York', 25),
        (2, 'Aman', 'japan', 35),
        (3, 'Ayush', 'New York', 40),
        (4, 'Pari', 'Mumbai', 30),
        (5, 'Kadu', 'Mumbai', 25),
    ]
    cursor.executemany("INSERT INTO customers VALUES (?, ?, ?, ?);", sample_data)
    conn.commit()
    return conn
```

```
/usr/local/lib/python3.11/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 as secre
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(
tokenizer_config.json: 100%  1.89k/1.89k [00:00<00:00, 87.0kB/s]
config.json: 100%  1.44k/1.44k [00:00<00:00, 83.3kB/s]
spiece.model: 100%  792k/792k [00:00<00:00, 4.49MB/s]
tokenizer.json: 100%  1.39M/1.39M [00:00<00:00, 9.76MB/s]
added_tokens.json: 100%  27.0/27.0 [00:00<00:00, 2.42kB/s]
special_tokens_map.json: 100%  1.79k/1.79k [00:00<00:00, 167kB/s]
pytorch_model.bin: 100%  11.4G/11.4G [01:36<00:00, 185MB/s]
model.safetensors: 11%  1.27G/11.4G [00:33<02:22, 71.3MB/s]
Error while downloading from https://cdn-lfs.hf.co/tscholak/cxmefzzi/104b1aadc4dc960a05d79ac7d9826e3407ad46705daa27927d2ff43d762b13ed?re
Trying to resume download...
WARNING:huggingface_hub.file_download:Error while downloading from https://cdn-lfs.hf.co/tscholak/cxmefzzi/104b1aadc4dc960a05d79ac7d9826
Trying to resume download...
model.safetensors: 25%  3.15G/12.7G [00:30<03:37, 43.9MB/s]
```

```
# Evaluation function
def evaluate(test_cases, schema, db_id):
    conn = create_sample_db()
    cursor = conn.cursor()
```

```
    cursor = conn.cursor()

    exact_match_count = 0
    execution_match_count = 0

    for i, case in enumerate(test_cases, 1):
        nl = case["nl"]
        expected_sql = case["expected_sql"].strip().lower()

        # Model prediction
        raw_predicted_sql = nl_to_sql(nl, db_id, schema)
        predicted_sql = clean_predicted_sql(raw_predicted_sql).strip().lower()

        # Exact match check
        is_exact_match = predicted_sql == expected_sql
        if is_exact_match:
            exact_match_count += 1

        # Execution accuracy check
        try:
            cursor.execute(expected_sql)
            expected_result = cursor.fetchall()
            cursor.execute(predicted_sql)
            predicted_result = cursor.fetchall()
            is_execution_match = expected_result == predicted_result
        except Exception as e:
            is_execution_match = False
            print(f" Execution Error in Test {i}: {e}")

        if is_execution_match:
            execution_match_count += 1

        # Show results
        print(f"\nTest Case {i}")
        print(f"NL: {nl}")
        print(f"Expected: {expected_sql}")
        print(f"Predicted: {predicted_sql}")
        print(f"✅ Exact Match: {is_exact_match}")
        print(f"✅ Execution Match: {is_execution_match}")

    total = len(test_cases)
    print("\n--- Evaluation Summary ---")
    print(f"✅ Exact Match Accuracy: {exact_match_count}/{total} = {exact_match_count / total:.2%}")
    print(f"✅ Execution Accuracy: {execution_match_count}/{total} = {execution_match_count / total:.2%}")
```

Evaluation Criteria

```
# Define test cases
test_cases = [
    {
        "nl": "What is the total number of customers from New York?",
        "expected_sql": "SELECT COUNT(*) FROM customers WHERE city = 'new york';"
    },
    {
        "nl": "List the names of customers older than 30.",
        "expected_sql": "SELECT name FROM customers WHERE age > 30;"
    },
    {
        "nl": "How many customers live in Mumbai?",
        "expected_sql": "SELECT COUNT(*) FROM customers WHERE city = 'mumbai';"
    }
]

# Schema and DB ID
schema = "customers : id (int), name (text), city (text), age (int)"
db_id = "customers_db"

# Run evaluation
evaluate(test_cases, schema, db_id)
```

```
Test Case 1
NL: What is the total number of customers from New York?
```

```
Expected: select count(*) from customers where city = 'new york';
Predicted: select count(*) from customers where city = 'new york city'
✅ Exact Match: False
✅ Execution Match: True

Test Case 2
NL: List the names of customers older than 30.
Expected: select name from customers where age > 30;
Predicted: select name from customers where age > 30
✅ Exact Match: False
✅ Execution Match: True

Test Case 3
NL: How many customers live in Mumbai?
Expected: select count(*) from customers where city = 'mumbai';
Predicted: select count(*) from customers where city = 'mumbai'
✅ Exact Match: False
✅ Execution Match: True

--- Evaluation Summary ---
✅ Exact Match Accuracy: 0/3 = 0.00%
✅ Execution Accuracy: 3/3 = 100.00%
```

New Section