What was done:
Pre Training Dataset is finished with included tag
Computed the metrics of F1-Score,Exact match,and Bleu

Code can be shown below:

```
from sklearn.metrics import f1_score
from nltk.translate.bleu_score import sentence_bleu
import nltk
nltk.download('punkt')
checkpoint = "Salesforce/codet5p-770m"
device = "cpu" # for GPU usage or "cpu" for CPU usage

tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = T5ForConditionalGeneration.from_pretrained(checkpoint).to(device)
# Read input text from Record2.txt
with open("/content/Record2.txt", "r") as f:
    input_text = f.read()

# Read expected output from Feature2.feature
with open("/content/Feature2.feature", "r") as f:
    expected_output = f.read()

inputs = tokenizer.encode(input_text, return_tensors="pt").to(device)
outputs = model.generate(inputs, max_length=20)
generated_output = tokenizer.decode(outputs[0], skip_special_tokens=True)

# Calculate F1-score
# Tokenize both outputs for comparison
expected_tokens = expected_output.split()
generated_tokens = generated_output.split()

# Pad shorter sequence with empty strings for equal length
max_len = max(len(expected_tokens), len(generated_tokens))
expected_tokens.extend([''] * (max_len - len(expected_tokens)))
generated_tokens.extend([''] * (max_len - len(generated_tokens)))

# Calculate F1-score using sklearn
f1 = f1_score(expected_tokens, generated_tokens, average='weighted', zero_division=0)

# Calculate BLEU score
reference = [expected_output.split()]  # Reference sentence as a list of words
candidate = generated_output.split()  # Candidate sentence as a list of words
bleu_score = sentence_bleu(reference, candidate)

# Calculate Exact Match
exact_match = 1 if generated_output == expected_output else 0

print("Input text:", input_text)
print("Generated Output:", generated_output)
print("Expected Output:", expected_output)
print("F1-score:", f1)
print("BLEU score:", bleu_score)
print("Exact Match:", exact_match)
```

Example output:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Input text {
<Scenario1>:Fixtures using a decorator
<Given1>:You can define `Django fixtures`_ using a function decorator. It is merely
a convenient way to keep fixtures close to your steps.
<Then1>:The decorator will
load the fixtures in the ``before_scenario``, as documented above.
}


Generated Output: <?php

namespace App\Http\Controllers;

use App\Http\
Expected Output: {
<Scenario1>:Load fixtures with the decorator
<Given1>:a step with a fixture decorator
<Then1>:the fixture should be loaded
}

F1-score: 0.0
BLEU score: 0
Exact Match: 0
```

Adjusted code to account for N number of records:

```python
from transformers import T5ForConditionalGeneration, AutoTokenizer
from sklearn.metrics import f1_score
from nltk.translate.bleu_score import sentence_bleu
import nltk
import os

nltk.download('punkt')
checkpoint = "Salesforce/codet5p-770m"
device = "cpu"

tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = T5ForConditionalGeneration.from_pretrained(checkpoint).to(device)

# Assuming files are in the current directory
file_prefix = "/content/"  # or specify a path, like "/path/to/files/"

# Get the list of feature files and extract the numbers
feature_files = [f for f in os.listdir(file_prefix) if f.startswith("Feature") and f.endswith(".feature")]
file_numbers = [int(f.split("Feature")[1].split(".")[0]) for f in feature_files]

# Iterate through files based on extracted numbers
for file_number in sorted(file_numbers):  # Ensures files are processed in order
    record_file = os.path.join(file_prefix, f"Record{file_number}.rst")  # Change to .rst
    feature_file = os.path.join(file_prefix, f"Feature{file_number}.feature")

    # Read input and expected output
    with open(record_file, "r") as f:
        input_text = f.read()
    with open(feature_file, "r") as f:
        expected_output = f.read()

    # Generate, evaluate and print as before (rest of the code remains the same)
    inputs = tokenizer.encode(input_text, return_tensors="pt").to(device)
    outputs = model.generate(inputs, max_length=20)
    generated_output = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Calculate F1-score using sklearn
    f1 = f1_score(expected_tokens, generated_tokens, average='weighted', zero_division=0)

    # Calculate BLEU score
    reference = [expected_output.split()]  # Reference sentence as a list of words
    candidate = generated_output.split()  # Candidate sentence as a list of words
    bleu_score = sentence_bleu(reference, candidate)

    # Calculate Exact Match
    exact_match = 1 if generated_output == expected_output else 0
    print("-----START OF RECORD",file_number,"-------")
    print("Input text:", input_text)
    print("Generated Output:", generated_output)
    print("Expected Output:", expected_output)
    print("F1-score:", f1)
    print("BLEU score:", bleu_score)
    print("Exact Match:", exact_match)
    print("-----END OF RECORD",file_number,"-------")
```

The first 5 records with respect to there features output can be shown in MetricReport.txt

Issues:
Currently either the format of the dataset or the fine tuning is not really providing anything good.