```
!pip install transformers
!pip install --upgrade transformers
!pip install transformers==4.11.3
```

```
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from tran
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.10
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from tr
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggin
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages
```

Resources ×

You are not subscribed. Learn more.

You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.

Manage sessions

Want more memory and disk space?

Upgrade to Colab Pro

Python 3 Google Compute Engine backend (GPU)

Showing resources from 2:16 PM to 2:49 PM

System RAM
4.4 / 12.7 GB

GPU RAM
11.3 / 15.0 GB

Disk
65.9 / 78.2 GB

```
  × Building wheel for tokenizers (pyproject.toml) did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
  Building wheel for tokenizers (pyproject.toml) ... error
  ERROR: Failed building wheel for tokenizers
  Building wheel for sacremoses (setup.py) ... done
  Created wheel for sacremoses: filename=sacremoses-0.0.53-py3-none-any.whl size=895239 sha25
  Stored in directory: /root/.cache/pip/wheels/00/24/97/a2ea5324f36bc626e1ea0267f33db6aa80d15
Successfully built sacremoses
Failed to build tokenizers
ERROR: Could not build wheels for tokenizers, which is required to install pyproject.toml-bas
```

```
from google.colab import drive
drive.mount('/content/drive')

%cd /content/drive/MyDrive/Temp03
```

```
Mounted at /content/drive
/content/drive/MyDrive/Temp03
```

```
#File Read Testing - ----------------------------------------------------Successfully Printed------
import json
import os

# Iterate over the files in the directory
filename = 'combined_dataset.json'
with open(filename, 'r') as file:
  data = json.load(file)

# Print the contents
print(f"Contents of {filename}:")
print(data)
```

```
Contents of combined_dataset.json:
{'classes': ['COURT', 'PETITONER', 'RESPONDENT', 'JUDGE', 'LAWYER', 'DATE', 'ORG', 'PROVISION',
```

```
# Extract classes/labels----------------------------------------------------Successfully Extracted-----
classes = data.get('classes', [])

# Display the results
print("Classes/Labels:")
print(classes)
```

```
Classes/Labels:
['COURT', 'PETITONER', 'RESPONDENT', 'JUDGE', 'LAWYER', 'DATE', 'ORG', 'PROVISION', 'PRECEDENT',
```

```
import json

# Step 1: Load the JSON data
filename = 'combined_dataset.json'
with open(filename, 'r', encoding='utf-8') as file:
    data = json.load(file)

# Step 2: Extract classes/labels
classes = data.get('classes', [])

# Step 3: Extract annotations and entities for each text sample
annotations_with_entities = []
ner_tags = set()  # Set to store all the NER tags

annotations = data.get('annotations', [])
for item in annotations:
    # Check if the item is a string (e.g., '--प्रतिवादीगण') and skip it
    if isinstance(item, str):
        continue

    text = item[0]  # The text sample
    entity_info = item[1]  # The annotations for the text sample

    # Extract entities from the entity_info
    entities = []
    for entity in entity_info.get('entities', []):
        start_index = entity[0]
        end_index = entity[1]
        entity_type = entity[2]
        entities.append((start_index, end_index, entity_type))

        ner_tags.add(entity_type)  # Add the entity type to the set of NER tags
```

```
    # Append the extracted data to the final list
    annotations_with_entities.append({'text': text, 'entities': entities})

# Display the results
print("Classes/Labels:")
print(classes)

print("\nNER Tags:")
print(ner_tags)

print("\nAnnotations with Entities:")
for item in annotations_with_entities:
    print(f"Text: {item['text']}")
    print(f"Entities: {item['entities']}")
    print()
```

```
Entities: []

Text: 06. बहस अंतिम उभय पक्षकारान् सुनी गयी।
Entities: []

Text:
Entities: []

Text: 07. दौराने बहस विद्वान अधिवक्ता वादी ने यह कथन किया कि विवादित
Entities: []

Text: भूखण्ड कुल 450 वर्गगज का वादी के स्वामित्व एवं कब्जे का भूखण्ड है जिसके
Entities: []

Text:
Entities: []

Text: न्यायालय : सिविल न्यायाधीश, फागी, जिला जयपुर (राज.)
Entities: [(28, 32, 'COURT'), (39, 44, 'DISTRICT'), (46, 49, 'STATE')]

Text:
Entities: []

Text: पीठासीन अधिकारी :- कोमल मण्डल,
Entities: [(19, 29, 'JUDGE')]

Text: (आर.जे.एस.)
Entities: []
```

MODEL

```
!pip install transformers==4.31.0 accelerate==0.20.3
```

```
Requirement already satisfied: transformers==4.31.0 in /usr/local/lib/python3.10/dist-packages
Collecting accelerate==0.20.3
  Downloading accelerate-0.20.3-py3-none-any.whl (227 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 227.6/227.6 kB 3.3 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transfo
Requirement already satisfied: huggingface-hub<1.0,>=0.14.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from tran
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from tran
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transfo
```

```
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from trans
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from acceleral
Requirement already satisfied: torch>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from acc
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingfa
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-pacl
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.6
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from to
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from req
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sym
Installing collected packages: accelerate
Successfully installed accelerate-0.20.3
```

```
from transformers import AutoTokenizer

model_checkpoint = "google/muril-base-cased"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
Downloading (…)okenizer_config.json: 100%  206/206 [00:00<00:00, 11.6kB/s]
Downloading (…)lve/main/config.json: 100%  411/411 [00:00<00:00, 28.2kB/s]
Downloading                                3.16M/3.16M [00:00<00:00
```

```
tokenizer.is_fast
```

```
True
```

```
import json
import torch
from torch.utils.data import Dataset, DataLoader
```

```
from transformers import AutoTokenizer, BertForTokenClassification, TrainingArguments, Trainer

# Step 1: Load the JSON data
filename = '/content/drive/MyDrive/Temp03/combined_dataset.json'  # Update the file path
with open(filename, 'r', encoding='utf-8') as file:
    data = json.load(file)


# Step 2: Extract classes/labels
classes = data.get('classes', [])

# Step 3: Extract annotations and entities for each text sample
annotations_with_entities = []
ner_tags = set()  # Set to store all the NER tags

annotations = data.get('annotations', [])
for item in annotations:
    # Check if the item is a string (e.g., '--प्रतिवादीगण') and skip it
    if isinstance(item, str):
        continue

    text = item[0]  # The text sample
    entity_info = item[1]  # The annotations for the text sample

    # Extract entities from the entity_info
    entities = []
    for entity in entity_info.get('entities', []):
        start_index = entity[0]
        end_index = entity[1]
        entity_type = entity[2]
        entities.append((start_index, end_index, entity_type))

        ner_tags.add(entity_type)  # Add the entity type to the set of NER tags

    # Append the extracted data to the final list
    annotations_with_entities.append({'text': text, 'entities': entities})

# Step 4: Tokenize the text samples using the specified model_checkpoint
model_checkpoint = "google/muril-base-cased"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)

# List to store the tokenized samples
tokenized_data = []
```

```
for item in annotations_with_entities:
    text = item['text']
    # Tokenize the text using the tokenizer
    inputs = tokenizer(text, return_offsets_mapping=True, truncation=True)

    # Find the token boundaries for each entity
    entities = item['entities']
    tokenized_entities = []
    for start, end, entity_type in entities:
        start_token = None
        end_token = None
        for i, (offset_start, offset_end) in enumerate(inputs['offset_mapping']):
            if offset_start == start:
                start_token = i
            if offset_end == end:
                end_token = i
                break
        if start_token is not None and end_token is not None:
            tokenized_entities.append((start_token, end_token, entity_type))

    # Append tokenized data to the list
    tokenized_data.append({'input_ids': inputs['input_ids'], 'entities': tokenized_entities})

    # Display tokenized data
for idx, item in enumerate(tokenized_data):
    print(f"Tokenized Text {idx}: {tokenizer.convert_ids_to_tokens(item['input_ids'])}")
    print(f"Tokenized Entities {idx}: {item['entities']}")
    print()
```

```
Tokenized Entities 1249: [(4, 8, 'DATE')]

Tokenized Text 1250: ['[CLS]', '[SEP]']
Tokenized Entities 1250: []

Tokenized Text 1251: ['[CLS]', '04', '.', 'यह', 'दीवानी', 'वाद', 'प्रकरण', 'पूर्व', '##िक', 'न्याया
Tokenized Entities 1251: []

Tokenized Text 1252: ['[CLS]', 'फ', '##ागी', ',', 'जिला', 'जयपुर', 'के', 'इस', 'न्यायालय', 'के'
Tokenized Entities 1252: []

Tokenized Text 1253: ['[CLS]', 'माननीय', 'जिला', 'एवं', 'सेशन', 'न्यायाधीश', ',', 'जयपुर', 'जिला',
Tokenized Entities 1253: [(10, 14, 'PRECEDENT')]

Tokenized Text 1254: ['[CLS]', '474', 'दिनांक', '04', '.', '42', '.', '2020', 'की', 'पालना', '
Tokenized Entities 1254: [(1, 7, 'PRECEDENT')]

Tokenized Text 1255: ['[CLS]', 'प्रकार', 'के', 'दीवानी', 'एवं', 'फौजदारी', 'प्रकरण', '##ों', 'को',
Tokenized Entities 1255: []

Tokenized Text 1256: ['[CLS]', 'की', 'अधिकार', '##िता', 'इस', 'न्यायालय', 'को', 'दिये', 'जाने',
Tokenized Entities 1256: []

Tokenized Text 1257: ['[CLS]', 'होने', 'पर', 'प्रकरण', 'दर्ज', 'रजिस्टर', 'किया', 'गया', '।', '[SEP
Tokenized Entities 1257: []

Tokenized Text 1258: ['[CLS]', '[SEP]']
Tokenized Entities 1258: []

Tokenized Text 1259: ['[CLS]', '02', '.', 'प्रकरण', 'के', 'संक्षिप्त', 'में', 'तथ्य', 'इस', 'प्रकार', '
Tokenized Entities 1259: []

Tokenized Text 1260: ['[CLS]', 'से', 'प्रति', '##वादी', '##गण', 'पूर', '##ण', 'व', '##गै', '##0',
Tokenized Entities 1260: []

Tokenized Text 1261: ['[CLS]', 'दिनांक', '29', '.', '07', '.', '204', '##3', 'को', 'न्यायालय', '
Tokenized Entities 1261: []

Tokenized Text 1262: ['[CLS]', '[SEP]']
Tokenized Entities 1262: []
```

```
from transformers import AutoTokenizer

model_checkpoint = "google/muril-base-cased"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
# Assuming you have the following data:
for x in annotations_with_entities:
    text = x['text']
    entities = x['entities']


    # Step 1: Tokenization
    tokens = tokenizer(text, return_offsets_mapping=True, is_split_into_words=False)
    words = tokens['input_ids']
    offsets = tokens['offset_mapping']

    # Step 2: Entity Conversion (BIO Scheme)
    entity_tags = ['O'] * len(words)
    for entity in entities:
        start, end, label = entity
        for i, (offset_start, offset_end) in enumerate(offsets):
            if offset_start >= start and offset_end <= end:
                entity_tags[i] = f"B-{label}" if i == start else f"I-{label}"

    print("Words:", words)
    print("Entity Tags:", entity_tags)
```

```
Entity Tags: ['O', 'O', 'O', 'B-LAWYER', 'I-LAWYER', 'I-LAWYER', 'I-LAWYER', 'O', 'O', 'O',
Words: [104, 126, 121, 2639, 5715, 2746, 120, 125, 1117, 2838, 1124, 1839, 9114, 1254, 492, 1
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
Words: [104, 127, 121, 2639, 5715, 2746, 120, 126, 1110, 34316, 1127, 80468, 29349, 492, 105]
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
Words: [104, 105]
Entity Tags: ['O', 'O']
Words: [104, 6589, 22016, 133, 1380, 121, 6118, 121, 51026, 105]
Entity Tags: ['O', 'O', 'O', 'O', 'I-DATE', 'I-DATE', 'I-DATE', 'I-DATE', 'I-DATE', 'O']
Words: [104, 105]
Entity Tags: ['O', 'O']
Words: [104, 7078, 121, 1233, 180965, 26439, 23868, 1554, 2430, 10286, 119, 10415, 2211, 3159
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
Words: [104, 452, 106530, 119, 2896, 16348, 1110, 1228, 10286, 1110, 1365, 1114, 1935, 191561
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
Words: [104, 68022, 2896, 1500, 182038, 15646, 119, 16348, 2896, 119, 16348, 1110, 6589, 8210
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'I-PRECEDENT', 'I-PRECEDENT',
Words: [104, 79308, 22016, 7078, 121, 4668, 121, 10584, 1117, 54635, 1114, 3891, 1554, 2430,
Entity Tags: ['B-PRECEDENT', 'I-PRECEDENT', 'I-PRECEDENT', 'I-PRECEDENT', 'I-PRECEDENT', 'I-P
Words: [104, 1768, 1110, 180965, 1500, 107608, 23868, 1325, 1125, 7599, 12353, 26910, 1500, 1
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
Words: [104, 1117, 3709, 9668, 1228, 10286, 1125, 17525, 2025, 1110, 1569, 105783, 83604, 715
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
Words: [104, 1694, 1154, 23868, 8060, 81698, 1219, 1258, 492, 105]
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
Words: [104, 105]
Entity Tags: ['O', 'O']
Words: [104, 7293, 121, 23868, 1110, 17064, 1114, 12468, 1228, 1768, 1124, 1115, 1187, 61090,
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
Words: [104, 1124, 2639, 5715, 28090, 84423, 2526, 461, 116921, 2211, 1110, 34316, 26439, 757
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
Words: [104, 22016, 2214, 121, 6796, 121, 35781, 2270, 1125, 10286, 1110, 25327, 1228, 47199,
Entity Tags: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
Words: [104, 105]
Entity Tags: ['O', 'O']
```

```
import torch
from transformers import Trainer
from torch.nn.utils.rnn import pad_sequence
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader, random_split
from transformers import AutoTokenizer, BertForTokenClassification, TrainingArguments, Trainer


# Assuming you have the tokenized_data and tokenizer from previous steps
```

```
# Step 5: Convert ner_tags to a list
ner_tags_list = list(ner_tags)

# Step 6: Create a custom dataset class
class NERDataset(Dataset):
    def __init__(self, tokenized_data, ner_tags):
        self.tokenized_data = tokenized_data
        self.ner_tags = ner_tags

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

    def __getitem__(self, idx):
        item = self.tokenized_data[idx]
        input_ids = torch.tensor(item["input_ids"])
        attention_mask = torch.ones_like(input_ids)  # Create attention mask with 1s
        entities = item["entities"]
        labels = self.get_labels(input_ids, entities)
        labels = torch.tensor(labels)

        return {
            "input_ids": input_ids,
            "attention_mask": attention_mask,
            "labels": labels,
        }

    def get_labels(self, input_ids, entities):
        labels = [-100] * len(input_ids)  # -100 is the ignore index for token classification
        for start, end, entity_type in entities:
            labels[start] = self.ner_tags.index(entity_type)
            for i in range(start + 1, end):
                labels[i] = self.ner_tags.index(entity_type)
        return labels

# Step 6: Create the dataset
dataset = NERDataset(tokenized_data, ner_tags_list)  # Pass tokenized_data and ner_tags_list only

# Step 7: Divide the dataset into training, validation, and testing sets
train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])
```

```
# Step 8: Define collate function to pad sequences
def collate_fn(batch):
    input_ids = [item["input_ids"] for item in batch]
    attention_mask = [item["attention_mask"] for item in batch]
    labels = [item["labels"] for item in batch]

    # Pad sequences to the same length within each batch
    input_ids = pad_sequence(input_ids, batch_first=True, padding_value=tokenizer.pad_token_id)
    attention_mask = pad_sequence(attention_mask, batch_first=True, padding_value=0)
    labels = pad_sequence(labels, batch_first=True, padding_value=-100)

    return {
        "input_ids": input_ids,
        "attention_mask": attention_mask,
        "labels": labels,
    }

# Step 9: Create data loaders for training, validation, and testing
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True, collate_fn=collate_fn)
val_loader = DataLoader(val_dataset, batch_size=16, collate_fn=collate_fn)
test_loader = DataLoader(test_dataset, batch_size=16, collate_fn=collate_fn)
```

```
# Step 9: Configure training arguments
output_dir = "./fine_tuned_model"
training_args = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=40,  # Adjust this as needed
    per_device_train_batch_size=5,  # Adjust this based on GPU memory
    save_steps=500,
    logging_steps=100,
    save_total_limit=2,
    overwrite_output_dir=True,
    evaluation_strategy="steps",  # Evaluate during training steps
    eval_steps=100,  # Evaluate every 100 steps
)
```

```
# Step 10: Create the Trainer and start fine-tuning
model = BertForTokenClassification.from_pretrained(model_checkpoint, num_labels=len(ner_tags))
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
```

```
    train_dataset=train_dataset,
    eval_dataset=val_dataset,  # Provide the validation dataset for evaluation
    data_collator=collate_fn,  # Use the collate_fn as data_collator
)
trainer.train()  # Start the fine-tuning process
```

```
Some weights of BertForTokenClassification were not initialized from the model checkpoint at goo
You should probably TRAIN this model on a down-stream task to be able to use it for predictions
```

[8080/8080 24:05, Epoch 40/40]

Step	Training Loss	Validation Loss
100	2.811300	2.745647
200	2.704500	2.638340
300	2.614200	2.553470
400	2.492000	2.431439
500	2.388100	2.316432
600	2.217600	2.173510
700	2.095500	2.042587
800	2.043900	1.906650
900	1.861300	1.800923
1000	1.774200	1.660209
1100	1.595800	1.559789
1200	1.459800	1.415816
1300	1.357700	1.259709
1400	1.220600	1.195144
1500	1.018500	1.070387
1600	1.051500	1.001329
1700	0.909300	1.029336
1800	0.828900	0.800223
1900	0.727200	0.848021
2000	0.588900	0.670090
2100	0.549200	0.798176
2200	0.557600	0.607877
2300	0.390800	0.598374
2400	0.397100	0.557465

2500	0.362300	0.533357
2600	0.403000	0.503298
2700	0.290400	0.588728
2800	0.254200	0.477784
2900	0.240500	0.449334
3000	0.230200	0.393085
3100	0.237000	0.423017
3200	0.190000	0.400292
3300	0.184100	0.389592
3400	0.176800	0.438111
3500	0.201300	0.422008
3600	0.142400	0.425604
3700	0.157700	0.372981
3800	0.124100	0.364714
3900	0.104300	0.352977
4000	0.097400	0.356857
4100	0.094000	0.347255
4200	0.085900	0.373184
4300	0.083500	0.359378
4400	0.071200	0.389566
4500	0.062500	0.358606
4600	0.065600	0.381471
4700	0.071500	0.379219
4800	0.067100	0.355723
4900	0.060800	0.363028
5000	0.053700	0.380574

5100	0.059100	0.364174
5200	0.055400	0.368461
5300	0.054700	0.425710
5400	0.039600	0.431550
5500	0.048700	0.368229
5600	0.042200	0.385739
5700	0.038800	0.343729
5800	0.046100	0.346694
5900	0.034800	0.343809
6000	0.042700	0.353000

```
# Step 12: Evaluate the model on the validation set
trainer.evaluate()

# Step 14: Save the pre-trained model
trainer.save_model("./ner_model")

# Optional: Save the tokenizer
tokenizer.save_pretrained("./ner_model")
```

[16/16 00:00]

```
('./ner_model/tokenizer_config.json',
 './ner_model/special_tokens_map.json',
 './ner_model/vocab.txt',
 './ner_model/added_tokens.json',
 './ner_model/tokenizer.json')
```

```
import torch

# Assuming you have already trained the model and it's loaded in a variable named 'model'

def evaluate_model(model, test_loader, device):
    model.eval()
    total_correct = 0
    total_samples = 0

    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch["input_ids"].to(device)
```

```
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["labels"].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
            _, predicted_labels = torch.max(outputs.logits, dim=2)

            # Calculate accuracy
            correct_mask = (labels != -100)  # Ignore index is -100, so we mask those tokens
            total_correct += torch.sum(predicted_labels[correct_mask] == labels[correct_mask])
            total_samples += torch.sum(correct_mask)

    accuracy = total_correct / total_samples
    return accuracy.item()

# Assuming you have already defined the 'device' variable
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Use the evaluate_model function to get accuracy on the test dataset
test_accuracy = evaluate_model(model, test_loader, device)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
Test Accuracy: 0.8195
```

```
import torch
from sklearn.metrics import precision_recall_fscore_support

# Assuming you have already trained the model and it's loaded in a variable named 'model'

def evaluate_model(model, test_loader, device):
    model.eval()
    total_correct = 0
    total_samples = 0
    all_predicted_labels = []
    all_true_labels = []

    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["labels"].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
```

```
            _, predicted_labels = torch.max(outputs.logits, dim=2)

            # Calculate accuracy
            correct_mask = (labels != -100)  # Ignore index is -100, so we mask those tokens
            total_correct += torch.sum(predicted_labels[correct_mask] == labels[correct_mask])
            total_samples += torch.sum(correct_mask)

            all_predicted_labels.extend(predicted_labels[correct_mask].cpu().numpy().tolist())
            all_true_labels.extend(labels[correct_mask].cpu().numpy().tolist())

    accuracy = total_correct / total_samples

    # Calculate Precision, Recall, and F1-score
    precision, recall, f1_score, _ = precision_recall_fscore_support(all_true_labels, all_predicted_l

    return accuracy.item(), precision, recall, f1_score

# Assuming you have already defined the 'device' variable and 'test_loader'

# Use the evaluate_model function to get accuracy, precision, recall, and f1-score on the test datase
test_accuracy, precision, recall, f1_score = evaluate_model(model, test_loader, device)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1_score:.4f}")
```

```
Test Accuracy: 0.8195
Precision: 0.8880
Recall: 0.8195
F1-score: 0.8299
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetri
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetri
  _warn_prf(average, modifier, msg_start, len(result))
```

✓ 1s completed at 2:46 PM