```
from faker import Faker
import random
import uuid
import pandas as pd

fake = Faker()

# Define event types
event_types = [
    "System Failure", "Fraudulent Activity", "Data Breach", "Unauthorized Access",
    "Service Outage", "Operational Error", "Compliance Violation", "Customer Dispute",
    "Payment Failure", "Insider Threat"
]

# Function to generate a fully unique event description
def generate_fully_unique_description(event_type):
    context = {
        "System Failure": f"A system failure was reported due to {fake.sentence()}. Engineers identified {fake.sentence()} "
                          f"which led to disruption. The team initiated {fake.sentence()} to restore functionality. "
                          f"Lessons learned included {fake.sentence()}. Future improvements will involve {fake.sentence()}.",

        "Fraudulent Activity": f"A fraudulent transaction was detected involving {fake.sentence()}. Investigation revealed {fake.sentence
                                f"which indicated an external breach. Customers were alerted and advised to {fake.sentence()}. "
                                f"Security teams recommended {fake.sentence()} to prevent recurrence.",

        "Data Breach": f"A data breach exposed customer information due to {fake.sentence()}. The vulnerability was traced to {fake.sente
                       f"Immediate actions were taken, including {fake.sentence()}. Compliance teams ensured {fake.sentence()} to minimiz

        "Unauthorized Access": f"An unauthorized access attempt was logged when {fake.sentence()}. The security system flagged the anomal
                                f"A forensic investigation found that {fake.sentence()}, leading to policy enhancements such as {fake.sent

        "Service Outage": f"A major service outage occurred due to {fake.sentence()}. Affected systems included {fake.sentence()}. "
                          f"Recovery steps involved {fake.sentence()}, and full functionality was restored after {fake.sentence()}.",

        "Operational Error": f"An operational error impacted several users when {fake.sentence()}. The root cause was identified as {fake
                              f"To prevent future occurrences, {fake.sentence()} was implemented, improving {fake.sentence()}.",

        "Compliance Violation": f"A compliance audit uncovered {fake.sentence()}. Investigation revealed that {fake.sentence()}, "
                                 f"necessitating corrective measures such as {fake.sentence()}. Future risk mitigation will include {fake.

        "Customer Dispute": f"A customer raised a dispute regarding {fake.sentence()}. Analysis showed that {fake.sentence()} contributed
                             f"Support teams resolved the matter by {fake.sentence()}, ensuring better transparency going forward.",

        "Payment Failure": f"A payment failure occurred when {fake.sentence()}. The failure was traced to {fake.sentence()}. "
                            f"Customers were notified, and the issue was resolved by {fake.sentence()}.",

        "Insider Threat": f"An insider threat incident was detected when {fake.sentence()}. Internal logs revealed {fake.sentence()}. "
                          f"Measures such as {fake.sentence()} were introduced to strengthen security against similar threats."
    }
    return context[event_type]

# Generate 5,000 records with fully unique descriptions
data_fully_unique = []
for _ in range(5000):
    event_type = random.choice(event_types)
    event_id = str(uuid.uuid4())[:8]  # Short unique identifier
    event_description = generate_fully_unique_description(event_type)
    data_fully_unique.append([event_id, event_type, event_description])

# Create DataFrame
df_fully_unique = pd.DataFrame(data_fully_unique, columns=["Event ID", "Event Type", "Event Description"])

# Save to CSV
file_path_fully_unique = "/content/sample_data/dummy_event_data_fully_unique.csv"
df_fully_unique.to_csv(file_path_fully_unique, index=False)

file_path_fully_unique
```

```
#!pip install faker
```

```
Collecting faker
  Downloading Faker-36.1.1-py3-none-any.whl.metadata (15 kB)
Requirement already satisfied: tzdata in /usr/local/lib/python3.11/dist-packages (from faker) (2025.1)
Downloading Faker-36.1.1-py3-none-any.whl (1.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 20.8 MB/s eta 0:00:00
Installing collected packages: faker
Successfully installed faker-36.1.1
```

```
import pandas as pd
import random
import uuid
from faker import Faker

# Initialize Faker
fake = Faker()

# Define event types
event_types = [
    "System Failure", "Fraudulent Activity", "Data Breach", "Unauthorized Access",
    "Service Outage", "Operational Error", "Compliance Violation", "Customer Dispute",
    "Payment Failure", "Insider Threat"
]

# Function to generate a unique event description
def generate_fully_unique_description(event_type):
    context = {
        "System Failure": f"A system failure was reported due to {fake.sentence()}. Engineers identified {fake.sentence()} "
                          f"which led to disruption. The team initiated {fake.sentence()} to restore functionality. "
                          f"Lessons learned included {fake.sentence()}. Future improvements will involve {fake.sentence()}.",

        "Fraudulent Activity": f"A fraudulent transaction was detected involving {fake.sentence()}. Investigation revealed {fake.senten
                                f"which indicated an external breach. Customers were alerted and advised to {fake.sentence()}. "
                                f"Security teams recommended {fake.sentence()} to prevent recurrence.",

        "Data Breach": f"A data breach exposed customer information due to {fake.sentence()}. The vulnerability was traced to {fake.sent
                        f"Immediate actions were taken, including {fake.sentence()}. Compliance teams ensured {fake.sentence()} to minim

        "Unauthorized Access": f"An unauthorized access attempt was logged when {fake.sentence()}. The security system flagged the anoma
                                f"A forensic investigation found that {fake.sentence()}, leading to policy enhancements such as {fake.se

        "Service Outage": f"A major service outage occurred due to {fake.sentence()}. Affected systems included {fake.sentence()}. "
                           f"Recovery steps involved {fake.sentence()}, and full functionality was restored after {fake.sentence()}.",

        "Operational Error": f"An operational error impacted several users when {fake.sentence()}. The root cause was identified as {fal
                              f"To prevent future occurrences, {fake.sentence()} was implemented, improving {fake.sentence()}.",

        "Compliance Violation": f"A compliance audit uncovered {fake.sentence()}. Investigation revealed that {fake.sentence()}, "
                                 f"necessitating corrective measures such as {fake.sentence()}. Future risk mitigation will include {fake

        "Customer Dispute": f"A customer raised a dispute regarding {fake.sentence()}. Analysis showed that {fake.sentence()} contribute
                             f"Support teams resolved the matter by {fake.sentence()}, ensuring better transparency going forward.",

        "Payment Failure": f"A payment failure occurred when {fake.sentence()}. The failure was traced to {fake.sentence()}. "
                            f"Customers were notified, and the issue was resolved by {fake.sentence()}.",

        "Insider Threat": f"An insider threat incident was detected when {fake.sentence()}. Internal logs revealed {fake.sentence()}. "
                           f"Measures such as {fake.sentence()} were introduced to strengthen security against similar threats."
    }
    return context[event_type]

# Generate NER tagging dataset
data_ner_tagging = []
num_records = 500  # Generate 500 records for this example

for i in range(num_records):
    event_type = random.choice(event_types)
    event_id = str(uuid.uuid4())[:8]  # Short unique identifier
    sentence = generate_fully_unique_description(event_type)

    # Fake named entities for tagging
    entities = {
        "SECURITY_ACTION": fake.word(),
        "ORG": fake.company(),
        "EVENT": event_type,
        "VULNERABILITY": fake.word(),
        "DATE": fake.date_this_decade().strftime("%Y-%m-%d"),
    }

    for entity_type, entity in entities.items():
        start_idx = sentence.find(entity)
        if start_idx != -1:
            end_idx = start_idx + len(entity)
            data_ner_tagging.append([event_id, sentence, entity, entity_type, start_idx, end_idx])

# Create DataFrame
df_ner_tagging = pd.DataFrame(data_ner_tagging, columns=["Sentence ID", "Sentence", "Entity", "Entity Type", "Start Index", "End Index"]

# Save to Excel
file_path_ner = "/content/sample_data/ner_tagging.xlsx"
df_ner_tagging.to_excel(file_path_ner, index=False)
```

```
file_path_ner
```

```
df_ner_tagging
```

	Sentence ID	Sentence	Entity	Entity Type	Start Index	End Index
0	7382a516	A major service outage occurred due to Skin re...	to	VULNERABILITY	36	38
1	f2e4c24a	A data breach exposed customer information due...	early	VULNERABILITY	216	221
2	96420d8e	A data breach exposed customer information due...	while	VULNERABILITY	240	245
3	a5854093	A customer raised a dispute regarding Consider...	by	VULNERABILITY	119	121
4	dd9c69ed	A payment failure occurred when Clear west nea...	into	SECURITY_ACTION	48	52
5	68525370	An insider threat incident was detected when R...	final	SECURITY_ACTION	55	60
6	8fefba0c	A major service outage occurred due to Believe...	form	SECURITY_ACTION	65	69
7	a01aa6af	A fraudulent transaction was detected involvin...	describe	SECURITY_ACTION	156	164
8	e4ee77ef	A major service outage occurred due to Hear re...	they	VULNERABILITY	176	180
9	2760f203	A compliance audit uncovered Walk science old ...	reveal	VULNERABILITY	87	93
10	cd9d29aa	A fraudulent transaction was detected involvin...	to	SECURITY_ACTION	191	193
11	4703875b	A customer raised a dispute regarding People d...	us	SECURITY_ACTION	3	5
12	ff6a7d22	An unauthorized access attempt was logged when...	Mrs	SECURITY_ACTION	120	123
13	3c037eb8	A fraudulent transaction was detected involvin...	stuff	VULNERABILITY	71	76
14	037cf08c	A data breach exposed customer information due...	various	VULNERABILITY	276	283
15	bb91daa7	An operational error impacted several users wh...	here	SECURITY_ACTION	74	78
16	009245b1	An unauthorized access attempt was logged when...	for	VULNERABILITY	192	195
17	18208553	A data breach exposed customer information due...	be	VULNERABILITY	167	169
18	b5862ca6	A system failure was reported due to Page air ...	discussion	VULNERABILITY	52	62
19	6f7129a7	A major service outage occurred due to Reduce ...	lose	SECURITY_ACTION	187	191
20	00b12175	A major service outage occurred due to Check a...	even	VULNERABILITY	222	226
21	ab6e85a3	A system failure was reported due to Data edge...	at	VULNERABILITY	38	40
22	9a51b4a2	A data breach exposed customer information due...	on	SECURITY_ACTION	40	42
23	0c59c6a0	An unauthorized access attempt was logged when...	all	VULNERABILITY	218	221
24	d12f541f	A compliance audit uncovered Cover off protect...	on	VULNERABILITY	80	82
25	cf524ae8	A system failure was reported due to First thr...	on	VULNERABILITY	69	71
26	59118c6b	A compliance audit uncovered Able fact cold fo...	as	VULNERABILITY	160	162
27	5d003e30	A customer raised a dispute regarding Wish mov...	team	VULNERABILITY	159	163
28	8a8ef7db	A major service outage occurred due to See exa...	out	VULNERABILITY	16	19
29	5b4f33bc	A system failure was reported due to Director ...	not	VULNERABILITY	50	53
30	d42e6ead	An insider threat incident was detected when E...	party	SECURITY_ACTION	205	210

```
import nltk

# Ensure necessary NLTK resources are available
nltk.download('punkt')

from nltk.tokenize import word_tokenize

# Sample sentences with entity tagging (manually tagged)
sentences = [
    ("1", "Microsoft acquired LinkedIn for $26 billion.", [("Microsoft", "B-ORG"), ("acquired", "O"), ("LinkedIn", "B-ORG"),
                                                          ("for", "O"), ("$26", "B-MONEY"), ("billion", "I-MONEY")]),

    ("2", "Amazon is planning to open a new office in New York.", [("Amazon", "B-ORG"), ("is", "O"), ("planning", "O"),
                                                                  ("to", "O"), ("open", "O"), ("a", "O"), ("new", "O"),
                                                                  ("office", "O"), ("in", "O"), ("New", "B-LOC"), ("York", "I-LOC")]),

    ("3", "Elon Musk founded SpaceX in 2002.", [("Elon", "B-PER"), ("Musk", "I-PER"), ("founded", "O"), ("SpaceX", "B-ORG"),
                                                ("in", "O"), ("2002", "B-DATE")]),

    ("4", "Google headquarters is in Mountain View, California.", [("Google", "B-ORG"), ("headquarters", "O"), ("is", "O"),
```

```
                                         ("in", "O"), ("Mountain", "B-LOC"), ("View", "I-LOC"),
                                         (",", "O"), ("California", "B-LOC")]),
]

# Convert data into a structured format
ner_data = []
for sent_id, sentence, tokens in sentences:
    for token, bio_tag in tokens:
        ner_data.append([sent_id, sentence, token, bio_tag])

# Create DataFrame
df_ner = pd.DataFrame(ner_data, columns=["Sentence ID", "Sentence", "Token", "BIO Tag"])

# Save to Excel
file_path_ner = "/content/sample_data/manual_ner_tagging.xlsx"
df_ner.to_excel(file_path_ner, index=False)

file_path_ner
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
df_ner
```

	Sentence ID	Sentence	Token	BIO Tag
0	1	Microsoft acquired LinkedIn for $26 billion.	Microsoft	B-ORG
1	1	Microsoft acquired LinkedIn for $26 billion.	acquired	O
2	1	Microsoft acquired LinkedIn for $26 billion.	LinkedIn	B-ORG
3	1	Microsoft acquired LinkedIn for $26 billion.	for	O
4	1	Microsoft acquired LinkedIn for $26 billion.	$26	B-MONEY
5	1	Microsoft acquired LinkedIn for $26 billion.	billion	I-MONEY
6	2	Amazon is planning to open a new office in New...	Amazon	B-ORG
7	2	Amazon is planning to open a new office in New...	is	O
8	2	Amazon is planning to open a new office in New...	planning	O
9	2	Amazon is planning to open a new office in New...	to	O
10	2	Amazon is planning to open a new office in New...	open	O
11	2	Amazon is planning to open a new office in New...	a	O
12	2	Amazon is planning to open a new office in New...	new	O
13	2	Amazon is planning to open a new office in New...	office	O
14	2	Amazon is planning to open a new office in New...	in	O
15	2	Amazon is planning to open a new office in New...	New	B-LOC
16	2	Amazon is planning to open a new office in New...	York	I-LOC
17	3	Elon Musk founded SpaceX in 2002.	Elon	B-PER
18	3	Elon Musk founded SpaceX in 2002.	Musk	I-PER
19	3	Elon Musk founded SpaceX in 2002.	founded	O
20	3	Elon Musk founded SpaceX in 2002.	SpaceX	B-ORG
21	3	Elon Musk founded SpaceX in 2002.	in	O
22	3	Elon Musk founded SpaceX in 2002.	2002	B-DATE
23	4	Google headquarters is in Mountain View, Calif...	Google	B-ORG
24	4	Google headquarters is in Mountain View, Calif...	headquarters	O
25	4	Google headquarters is in Mountain View, Calif...	is	O
26	4	Google headquarters is in Mountain View, Calif...	in	O
27	4	Google headquarters is in Mountain View, Calif...	Mountain	B-LOC
28	4	Google headquarters is in Mountain View, Calif...	View	I-LOC
29	4	Google headquarters is in Mountain View, Calif...	,	O
30	4	Google headquarters is in Mountain View, Calif...	California	B-LOC

```
# import pandas as pd
```

```
# # Replace 'file_path' with the actual path of your Excel file
# file_path = "path_to_your_file.xlsx"
# df = pd.read_excel(file_path)

# # Display the first few rows of the DataFrame
# print(df.head())



# Yes, you can use the BERT Base Uncased model for Named Entity Recognition (NER) tagging! Here's how it works:

# 1. Pre-trained BERT Model
# BERT (Bidirectional Encoder Representations from Transformers) has been pre-trained on large text corpora and fine-tuned for specific

# 2. Fine-Tuning BERT for NER
# While you can use the pre-trained BERT Base Uncased directly, fine-tuning it on a labeled NER dataset is the best approach for achiev

# 3. Using Hugging Face Transformers for NER
# The Hugging Face Transformers library makes it easy to load pre-trained models like BERT and fine-tune them for tasks like NER. Here’s

# Code Example: Using BERT for NER with Hugging Face Transformers


from transformers import BertTokenizer, BertForTokenClassification
from transformers import pipeline

# Load pre-trained BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")
model = BertForTokenClassification.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

# Initialize the NER pipeline
nlp_ner = pipeline("ner", model=model, tokenizer=tokenizer)

# Sample text
text = "Apple is looking at buying U.K. startup for $1 billion"

# Perform NER
ner_results = nlp_ner(text)

# Print results
for entity in ner_results:
    print(f"Entity: {entity['word']}, Label: {entity['entity']}, Confidence: {entity['score']:.4f}")
```

```
/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
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%  60.0/60.0 [00:00<00:00, 2.01kB/s]
vocab.txt: 100%  213k/213k [00:00<00:00, 608kB/s]
config.json: 100%  998/998 [00:00<00:00, 25.4kB/s]
model.safetensors: 100%  1.33G/1.33G [00:14<00:00, 161MB/s]
Some weights of the model checkpoint at dbmdz/bert-large-cased-finetuned-conll03-english were not used when initializing BertForToke
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exact
Device set to use cpu
Entity: Apple, Label: I-ORG, Confidence: 0.9990
Entity: U, Label: I-LOC, Confidence: 0.9997
Entity: K, Label: I-LOC, Confidence: 0.9979
```

```
# I understand now! You want the data grouped by each sentence so that all tokens of a sentence appear in consecutive rows. Here's how w

# Group the data by Sentence ID (to keep sentences together).
# Include each token on a new row under each sentence.
# Each token will have a corresponding BIO tag.
# Let me process the data for you in this structure. I'll group by Sentence ID and align each token under its respective sentence. Let's


# Grouping and displaying the tokens with their BIO tag for each sentence
df_grouped_ner = df_ner.groupby('Sentence ID').apply(lambda x: x[['Token', 'BIO Tag']]).reset_index()

# Display the resulting DataFrame
df_grouped_ner.head()
```

```
<ipython-input-17-1e8bf32c1441>:2: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is dep
  df_grouped_ner = df_ner.groupby('Sentence ID').apply(lambda x: x[['Token', 'BIO Tag']]).reset_index()
```

	Sentence ID	level_1	Token	BIO Tag
0	1	0	Microsoft	B-ORG
1	1	1	acquired	O
2	1	2	LinkedIn	B-ORG
3	1	3	for	O
4	1	4	$26	B-MONEY

```
import pandas as pd

# Assuming you have already loaded your data into a DataFrame `df_ner`

# Group by 'Sentence ID' and reset index
df_grouped = df_ner.groupby('Sentence ID').apply(lambda x: x[['Token', 'BIO Tag']]).reset_index(drop=True)

# Display the resulting DataFrame
print(df_grouped)
```

```
           Token  BIO Tag
0      Microsoft    B-ORG
1       acquired        O
2       LinkedIn    B-ORG
3            for        O
4            $26  B-MONEY
5        billion  I-MONEY
6         Amazon    B-ORG
7             is        O
8       planning        O
9             to        O
10          open        O
11             a        O
12           new        O
13        office        O
14            in        O
15           New    B-LOC
16          York    I-LOC
17          Elon    B-PER
18          Musk    I-PER
19       founded        O
20        SpaceX    B-ORG
21            in        O
22          2002   B-DATE
23        Google    B-ORG
24  headquarters        O
25            is        O
26            in        O
27      Mountain    B-LOC
28          View    I-LOC
29             ,        O
30    California    B-LOC
<ipython-input-18-0bcb32e753d9>:6: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is dep
  df_grouped = df_ner.groupby('Sentence ID').apply(lambda x: x[['Token', 'BIO Tag']]).reset_index(drop=True)
```

```
import pandas as pd

# Assuming df_ner is your DataFrame with Token and Sentence ID columns

# Group by 'Sentence ID', combine all tokens for each sentence, and reset index
df_combined = df_ner.groupby('Sentence ID').apply(lambda x: ' '.join(x['Token'])).reset_index(name='Sentence')

# Display the resulting DataFrame
print(df_combined)
```

```
   Sentence ID                                     Sentence
0            1        Microsoft acquired LinkedIn for $26 billion
1            2  Amazon is planning to open a new office in New...
2            3                   Elon Musk founded SpaceX in 2002
3            4  Google headquarters is in Mountain View , Cali...
<ipython-input-19-433b56dc8a45>:6: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is dep
  df_combined = df_ner.groupby('Sentence ID').apply(lambda x: ' '.join(x['Token'])).reset_index(name='Sentence')
```

```
# whcih bert model has capability for ner tagging,token ids,
```

```
# ChatGPT said:
# For NER tagging and generating token IDs, several pre-trained BERT models are available, and Hugging Face provides various fine-tuned

# 1. BERT-based Models for NER
# 1. BERT-base uncased / cased (pre-trained models)
# Model Name: bert-base-uncased (or bert-base-cased for case-sensitive tasks)
# Capabilities:
# Tokenization: Can generate token IDs for input sentences.
# NER: You can fine-tune this model for Named Entity Recognition (NER) using your custom data.
# Example:
# python
# Copy
# Edit
# from transformers import BertTokenizer, BertForTokenClassification

# tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
# model = BertForTokenClassification.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

# # Tokenize input text and get token IDs
# inputs = tokenizer("Apple acquired LinkedIn", return_tensors="pt", padding=True, truncation=True)
# token_ids = inputs['input_ids']
# 2. Fine-tuned BERT Models for NER
# Model Name: dbmdz/bert-large-cased-finetuned-conll03-english or dbmdz/bert-base-cased-finetuned-conll03-english

# Capabilities:

# NER: Fine-tuned for NER tasks on the CoNLL-03 dataset.
# Tokenization: Generates token IDs specific to NER tagging.
# Example:

# python
# Copy
# Edit
# from transformers import BertTokenizer, BertForTokenClassification
# from transformers import pipeline

# tokenizer = BertTokenizer.from_pretrained("dbmdz/bert-base-cased-finetuned-conll03-english")
# model = BertForTokenClassification.from_pretrained("dbmdz/bert-base-cased-finetuned-conll03-english")

# # Example sentence
# sentence = "Apple is looking to acquire LinkedIn for $26 billion."

# # Tokenize and generate token IDs
# inputs = tokenizer(sentence, return_tensors="pt", padding=True, truncation=True)
# token_ids = inputs['input_ids']

# # Perform NER
# nlp_ner = pipeline("ner", model=model, tokenizer=tokenizer)
# ner_results = nlp_ner(sentence)
# 3. BERT-based Multilingual Model
# Model Name: bert-base-multilingual-cased (useful if working with multiple languages)
# Capabilities:
# Can generate token IDs for sentences in various languages.
# Fine-tuning for NER can be done with a multilingual NER dataset.
# Example:
# python
# Copy
# Edit
# tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-cased")
# model = BertForTokenClassification.from_pretrained("dbmdz/bert-base-multilingual-cased-finetuned-conll03-english")

# sentence = "Microsoft acquired LinkedIn"
# inputs = tokenizer(sentence, return_tensors="pt")
# token_ids = inputs['input_ids']
# 2. Specialized Models for NER
# You can also use specialized models that are specifically fine-tuned for NER tasks. These include:

# Model Name: dbmdz/bert-large-cased-finetuned-conll03-english
# Fine-tuned on the CoNLL-03 dataset for NER and includes predefined tags like Person (PER), Organization (ORG), Location (LOC), etc.
# Model Name: bert-base-uncased (fine-tuned on your dataset)
# The base model can be fine-tuned for NER tasks on your specific data using Hugging Face's tools.
# How to Use BERT for Token IDs and NER Tagging:
# Here is an example combining tokenization, token IDs, and NER tagging using a fine-tuned BERT model:

# python
# Copy
# Edit
# from transformers import BertTokenizer, BertForTokenClassification
# from transformers import pipeline

# # Load the pre-trained model and tokenizer
```

```
# model_name = "dbmdz/bert-large-cased-finetuned-conll03-english"
# tokenizer = BertTokenizer.from_pretrained(model_name)
# model = BertForTokenClassification.from_pretrained(model_name)

# # Example sentence
# sentence = "Apple acquired LinkedIn for $26 billion."

# # Tokenize the sentence and get token IDs
# inputs = tokenizer(sentence, return_tensors="pt", padding=True, truncation=True)
# token_ids = inputs['input_ids']

# # Perform NER using the model
# nlp_ner = pipeline("ner", model=model, tokenizer=tokenizer)
# ner_results = nlp_ner(sentence)

# # Output token IDs and NER results
# print("Token IDs:", token_ids)
# print("NER Results:", ner_results)
# Output Example:
# Token IDs: The output will show the token IDs corresponding to each token in the input sentence.
# NER Results: The output will show the entity, label, and confidence score for each recognized entity (e.g., "Apple" as ORG, "LinkedIn'
# Conclusion:
# You can use models like bert-base-uncased for NER tagging and token IDs.
# Fine-tuned models like dbmdz/bert-base-cased-finetuned-conll03-english are pre-trained on NER tasks and can be used directly.
# Hugging Face provides an easy-to-use interface for tokenization and model inference.
# Would you like a more detailed explanation or sample code?


# To build a Named Entity Recognition (NER) model that leverages BERT's capabilities along with the manually created tagging Excel data,

# Overview of Steps:
# Preprocess the Data:

# Read the manually tagged data (e.g., from an Excel file) into a DataFrame.
# Tokenize the data using a BERT tokenizer.
# Convert the manually labeled tags into BIO format (Begin, Inside, Outside) and prepare it for training.
# Fine-Tune a BERT Model for NER:

# Use a pre-trained BERT model (e.g., bert-base-uncased) for fine-tuning with the manually tagged data.
# Train the model on your custom dataset to recognize entities.
# Evaluate the Model:

# Use the model to predict NER on unseen data.
# Detailed Steps to Build the NER Model:
# Step 1: Load the Manually Tagged Data
# Read the Excel File: First, you need to read the data from the Excel file, which contains the manually tagged entities (in BIO format)


import pandas as pd

# Load the manually tagged data from an Excel file (adjust the path as necessary)
df_ner = pd.read_excel('/content/sample_data/manual_ner_tagging.xlsx')

# Display the first few rows of the data
print(df_ner.head())
```

```
   Sentence ID                                  Sentence      Token  \
0            1  Microsoft acquired LinkedIn for $26 billion.  Microsoft
1            1  Microsoft acquired LinkedIn for $26 billion.   acquired
2            1  Microsoft acquired LinkedIn for $26 billion.   LinkedIn
3            1  Microsoft acquired LinkedIn for $26 billion.        for
4            1  Microsoft acquired LinkedIn for $26 billion.        $26

   BIO Tag
0    B-ORG
1        O
2    B-ORG
3        O
4  B-MONEY
```

```
# Assuming the Excel data has columns like:

# Sentence ID: Unique ID for each sentence.
# Token: The individual token/word.
# BIO Tag: The NER label for each token (e.g., B-ORG, I-ORG, O).
# Step 2: Preprocess the Data
# Convert the Data to the BERT Format:
# BERT's tokenizer expects data in a specific format. We'll group the data by Sentence ID and tokenize it.


from transformers import BertTokenizer

# Load the BERT tokenizer
```

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Function to prepare data for BERT (tokenization and BIO conversion)
def prepare_data_for_bert(df):
    sentences = []
    labels = []

    for sentence_id, group in df.groupby('Sentence ID'):
        sentence = group['Token'].tolist()
        bio_tags = group['BIO Tag'].tolist()

        # Tokenize sentence using BERT tokenizer
        encodings = tokenizer(sentence, truncation=True, padding='max_length', is_split_into_words=True)

        # Ensure the labels match the number of tokens
        word_labels = bio_tags

        sentences.append(encodings)
        labels.append(word_labels)

    return sentences, labels

# Prepare the data
sentences, labels = prepare_data_for_bert(df_ner)
```

```
tokenizer_config.json:  100%    48.0/48.0  [00:00<00:00,  3.21kB/s]
vocab.txt:  100%    232k/232k  [00:00<00:00,  676kB/s]
tokenizer.json:  100%    466k/466k  [00:00<00:00,  1.36MB/s]
config.json:  100%    570/570  [00:00<00:00,  44.4kB/s]
```

```
# Step 3: Fine-Tune the BERT Model
# Fine-Tune BERT for NER: We'll fine-tune the pre-trained BERT model on the manually tagged data using Hugging Face's Trainer API.


from transformers import BertForTokenClassification, Trainer, TrainingArguments
import torch
from torch.utils.data import Dataset, DataLoader

# Create a custom dataset class for the NER task
class NERDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

# Convert sentences and labels into a dataset
train_dataset = NERDataset(sentences, labels)

# Load the pre-trained BERT model for token classification
model = BertForTokenClassification.from_pretrained('bert-base-uncased', num_labels=len(set(df_ner['BIO Tag'])))

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    logging_dir='./logs',
)

# Set up the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
)

# Train the model
trainer.train()
```

```
model.safetensors: 100%                                    440M/440M [00:02<00:00, 257MB/s]
Some weights of BertForTokenClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initial
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter:
```

```
# Step 4: Evaluate the Model
# Test the Model: After fine-tuning, you can evaluate the model's performance on new data or use it for inference.


# Test with a new sentence
test_sentence = "Apple is acquiring LinkedIn."

# Tokenize the test sentence
test_encodings = tokenizer(test_sentence, return_tensors="pt", truncation=True, padding=True, is_split_into_words=True)

# Get predictions
outputs = model(**test_encodings)
predictions = torch.argmax(outputs.logits, dim=2)

# Convert predicted token IDs to BIO tags
predicted_labels = [model.config.id2label[label.item()] for label in predictions[0]]

# Print the results
tokens = test_sentence.split()
for token, label in zip(tokens, predicted_labels):
    print(f"Token: {token}, Predicted Label: {label}")



# Final Output:
# Tokens: The individual words or subwords that BERT processes.
# Predicted Labels: The entity class for each token (e.g., B-ORG, I-ORG, O).
# Considerations:
# BIO Labeling: Make sure your manually tagged data follows the BIO format (B- for beginning, I- for inside, O- for outside).
# Model Hyperparameters: You can adjust the batch size and learning rate during fine-tuning based on your dataset.
# Fine-Tuning Duration: Fine-tuning could take a while depending on your hardware, so make sure to adjust the number of epochs and batch
# Conclusion:
# This workflow enables you to fine-tune a BERT-based NER model using both pre-trained BERT capabilities and your manually created taggi

# Would you like additional details or code improvements


# I see! You want to pass an aggregated sentence file, where each sentence is in a file (such as a CSV or Excel), and then process that

# Steps:
# Read the input file (which contains the aggregated sentences).
# Tokenize each sentence and generate the tokenization details.
# Create a detailed tokenization report and store it in a structured format (like a CSV or Excel).
# Let's assume you have an Excel file (sentences.xlsx) with one column for sentences. Here's how you can proceed:


import pandas as pd
from transformers import BertTokenizer

# Initialize the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Read your input sentences file (replace 'sentences.xlsx' with your actual file path)
file_path = '/path/to/your/sentences.xlsx'
df = pd.read_excel(file_path)

# Assuming the column name containing sentences is 'Sentence'
sentences = df['Sentence'].tolist()

# Function to generate a detailed tokenization report
def generate_tokenization_report(sentences):
    report = []

    for sentence in sentences:
        # Tokenize the sentence and capture information
        encoding = tokenizer(sentence, return_tensors='pt', padding=True, truncation=True, is_split_into_words=False)

        # Tokenized words and their corresponding IDs
        tokens = tokenizer.convert_ids_to_tokens(encoding['input_ids'][0])

        # Number of tokens
```

```
        num_tokens = len(tokens)

        # Starting and ending tokens
        start_token = tokens[0]  # The start token of the sentence
        end_token = tokens[-1]   # The end token of the sentence

        # Report structure
        sentence_report = {
            'Original Sentence': sentence,
            'Start Token': start_token,
            'End Token': end_token,
            'Number of Tokens': num_tokens,
            'Tokens & Token IDs': [(token, tokenizer.convert_tokens_to_ids(token)) for token in tokens]
        }

        report.append(sentence_report)

    return report

# Generate the tokenization report
tokenization_report = generate_tokenization_report(sentences)

# Convert the report to a DataFrame for better visualization
report_data = []
for report in tokenization_report:
    for token, token_id in report['Tokens & Token IDs']:
        report_data.append({
            'Original Sentence': report['Original Sentence'],
            'Start Token': report['Start Token'],
            'End Token': report['End Token'],
            'Number of Tokens': report['Number of Tokens'],
            'Token': token,
            'Token ID': token_id
        })

df_report = pd.DataFrame(report_data)

# Display the report
print(df_report)

# If you want to save it to Excel:
output_file = '/content/sample_data/tokenization_report.xlsx'
df_report.to_excel(output_file, index=False)

output_file



# Explanation:
# Reading the Input File:

# The file sentences.xlsx is read using pd.read_excel(), and it assumes that the sentences are stored in a column named 'Sentence'. Modi
# The sentences are stored in the sentences list for processing.
# Tokenization Process:

# The sentences are tokenized using the BERT tokenizer. The generate_tokenization_report function processes each sentence, captures toke
# Creating the Report:

# The df_report DataFrame is created from the tokenization_report, which includes all the tokens and their corresponding token IDs, alor
# Saving the Report:

# The final tokenization report is saved as an Excel file at the path '/mnt/data/tokenization_report.xlsx'. You can download it from the
# Output:
# The resulting Excel file (tokenization_report.xlsx) will look something like this:

# Original Sentence Start Token End Token   Number of Tokens    Token   Token ID
# Microsoft acquired LinkedIn for $26 billion.  [CLS]   [SEP]   9   [CLS]   101
# Microsoft acquired LinkedIn for $26 billion.  [CLS]   [SEP]   9   microsoft   2046
# Microsoft acquired LinkedIn for $26 billion.  [CLS]   [SEP]   9   acquired    13651
# Microsoft acquired LinkedIn for $26 billion.  [CLS]   [SEP]   9   linked  1037
# ...   ... ... ... ... ...
# Customization:
# Sentence Column: Change the column name ('Sentence') to match the actual column in your file.
# File Format: If you have a CSV file instead of Excel, use pd.read_csv() instead of pd.read_excel().


If you prefer to avoid using an API key for accessing pre-trained models, we can take an alternative approach by using local models and

Here's how you can achieve this:

Alternative Approach: Fine-Tune BERT for NER Locally
We'll follow the same general approach, but instead of using an API, we will rely on local pre-trained models and Hugging Face's transfo
```

```
Steps:
Prepare the Data:

Read the manually tagged data (from Excel or CSV) into a pandas DataFrame.
Convert the data into the BIO format for NER.
Set Up the Local Model:

Use a pre-trained BERT model stored locally.
Use the Hugging Face library to load the model and fine-tune it on your local machine.
Fine-Tune the Model:

Fine-tune the model with your manually tagged data on your local environment.
Evaluate the Model:

Evaluate the model on test data to check its performance.
Step-by-Step Code Implementation:
Step 1: Preprocess the Data
We will use the manually tagged BIO format from the Excel file. Here is the code to prepare your data:

python
Copy
Edit
import pandas as pd
from transformers import BertTokenizer
import torch
from torch.utils.data import Dataset, DataLoader

# Load the manually tagged data from an Excel file (adjust the path as necessary)
df_ner = pd.read_excel('path_to_manually_tagged_data.xlsx')

# Define the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Function to prepare data for BERT (tokenization and BIO conversion)
def prepare_data_for_bert(df):
    sentences = []
    labels = []

    for sentence_id, group in df.groupby('Sentence ID'):
        sentence = group['Token'].tolist()
        bio_tags = group['BIO Tag'].tolist()

        # Tokenize sentence using BERT tokenizer
        encodings = tokenizer(sentence, truncation=True, padding='max_length', is_split_into_words=True)

        # Ensure the labels match the number of tokens
        word_labels = bio_tags

        sentences.append(encodings)
        labels.append(word_labels)

    return sentences, labels

# Prepare the data
sentences, labels = prepare_data_for_bert(df_ner)
Step 2: Set Up the Local Model
Load a pre-trained BERT model locally, without using an API key:

python
Copy
Edit
from transformers import BertForTokenClassification

# Load a local pre-trained BERT model (using 'bert-base-uncased' as an example)
model = BertForTokenClassification.from_pretrained('bert-base-uncased', num_labels=len(set(df_ner['BIO Tag'])))
Make sure you have the transformers and torch libraries installed locally.

Step 3: Fine-Tune the Model
Use PyTorch to fine-tune the model locally on your dataset. We will define a custom Dataset for NER and then set up the Trainer for fine

python
Copy
Edit
from transformers import Trainer, TrainingArguments

# Define a custom dataset class
class NERDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
```

```
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

# Create the dataset for training
train_dataset = NERDataset(sentences, labels)

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',  # Directory for storing results
    num_train_epochs=3,      # Number of training epochs
    per_device_train_batch_size=8,  # Batch size
    per_device_eval_batch_size=8,   # Batch size during evaluation
    logging_dir='./logs',          # Directory for logs
    evaluation_strategy="epoch",   # Evaluate at the end of each epoch
)

# Set up the Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
)

# Fine-tune the model
trainer.train()
Step 4: Evaluate the Model
Once fine-tuning is complete, you can test the model with a new sentence and check its performance.

python
Copy
Edit
# Test sentence for NER prediction
test_sentence = "Apple is acquiring LinkedIn."

# Tokenize the test sentence
test_encodings = tokenizer(test_sentence, return_tensors="pt", truncation=True, padding=True, is_split_into_words=True)

# Get predictions
outputs = model(**test_encodings)
predictions = torch.argmax(outputs.logits, dim=2)

# Convert predicted token IDs to BIO tags
predicted_labels = [model.config.id2label[label.item()] for label in predictions[0]]

# Print results
tokens = test_sentence.split()
for token, label in zip(tokens, predicted_labels):
    print(f"Token: {token}, Predicted Label: {label}")
Summary of the Approach:
Local Model: You load a pre-trained BERT model directly from the Hugging Face transformers library, without needing an API key.
Training: You fine-tune the model using your custom NER dataset, which includes manually tagged tokens in BIO format.
Evaluation: You can test the model on new sentences to see its entity recognition capabilities.
This approach avoids the need for API keys and allows you to fine-tune the model completely within your local environment.

Let me know if you need help setting up the environment or any specific part of the process!


full code


import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertForTokenClassification, Trainer, TrainingArguments
from transformers import pipeline
from datasets import Dataset, DatasetDict
from seqeval.metrics import classification_report
import torch
import numpy as np

# Load the manually tagged NER data (Excel file)
file_path = '/path/to/your/ner_tagged_data.xlsx'  # Modify this to your file's location
df = pd.read_excel(file_path)

# BERT tokenizer for tokenizing sentences
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Function to encode the data in BERT format
def encode_data(df, tokenizer):
    # Prepare sentence, tokens and labels
```

```
        sentences = df['Sentence'].tolist()
        tokens = df['Token'].tolist()
        bio_tags = df['BIO Tag'].tolist()

        # Encoding the sentences and the corresponding BIO tags
        encodings = tokenizer(sentences, padding=True, truncation=True, is_split_into_words=True, return_tensors='pt')
        labels = []

        # Create BIO labels
        for sentence, token, bio_tag in zip(sentences, tokens, bio_tags):
            # Create label ids from the BIO tags
            sentence_labels = []
            for word, bio in zip(sentence.split(), bio_tag.split()):
                sentence_labels.append(bio)  # This should correspond to the BIO label for each word

            labels.append(sentence_labels)

        return encodings, labels

    # Prepare the data for NER fine-tuning
    encodings, labels = encode_data(df, tokenizer)

    # Convert to Hugging Face dataset
    dataset = Dataset.from_dict({'input_ids': encodings['input_ids'].numpy(),
                                 'attention_mask': encodings['attention_mask'].numpy(),
                                 'labels': labels})

    # Split the dataset into training, validation, and test sets
    train_test_split = dataset.train_test_split(test_size=0.1)
    train_val_split = train_test_split['train'].train_test_split(test_size=0.1)

    train_dataset = train_val_split['train']
    val_dataset = train_val_split['test']
    test_dataset = train_test_split['test']

    # Initialize BERT model for token classification (NER)
    model = BertForTokenClassification.from_pretrained('bert-base-uncased', num_labels=len(set(df['BIO Tag'].unique())))

    # Define training arguments
    training_args = TrainingArguments(
        output_dir='./results',
        evaluation_strategy="epoch",
        learning_rate=2e-5,
        per_device_train_batch_size=16,
        per_device_eval_batch_size=64,
        num_train_epochs=3,
        weight_decay=0.01,
        logging_dir='./logs',
        logging_steps=10,
        save_steps=500,
        save_total_limit=2,
        load_best_model_at_end=True,
    )

    # Initialize the Trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=val_dataset,
        tokenizer=tokenizer,
        data_collator=lambda data: {'input_ids': torch.stack([f[0] for f in data]),
                                    'attention_mask': torch.stack([f[1] for f in data]),
                                    'labels': torch.stack([f[2] for f in data])},
    )

    # Fine-tune the model
    trainer.train()

    # Evaluate the model on the test set
    test_results = trainer.evaluate(test_dataset)

    # Save the fine-tuned model
    model.save_pretrained('fine_tuned_ner_model')
    tokenizer.save_pretrained('fine_tuned_ner_model')

    # Evaluate the model and print classification report
    predictions, labels, _ = trainer.predict(test_dataset)
    predictions = np.argmax(predictions, axis=2)

    # Convert predictions to BIO labels
    predictions = [[model.config.id2label[label] for label in prediction] for prediction in predictions]
```

```
labels = [[model.config.id2label[label] for label in label_set] for label_set in labels]

# Print classification report (precision, recall, F1-score)
print(classification_report(labels, predictions))

# Save the evaluation results
with open('evaluation_results.txt', 'w') as f:
    f.write(str(test_results))

# Model evaluation and saving results
print("Model fine-tuned and saved successfully.")



The model bert-base-uncased can be a good starting point for Named Entity Recognition (NER) tasks, especially if you have a relatively s

When bert-base-uncased is Sufficient:
General Purpose NER: If your NER task involves general entities (like organizations, people, locations, etc.) and is not heavily domain-
Pre-trained on Large Corpora: bert-base-uncased has been pre-trained on a vast amount of data (BooksCorpus and English Wikipedia), so it
When You Might Need a More Specialized Model:
Domain-Specific Entities: If your data involves specialized terminology (e.g., legal entities, medical entities, or financial terms), a

Language Variety: If your data involves slang, social media text, or non-standard writing (e.g., tweets, product reviews), the model mig

Considerations for Fine-Tuning bert-base-uncased for NER:
Fine-Tuning Process: As you've planned, fine-tuning the model with your labeled data is essential. The model starts with general knowle

Performance Expectations: During the fine-tuning process, make sure to evaluate metrics like precision, recall, and F1 score to assess t

Alternative Models to Consider:
bert-base-cased: The cased version of BERT may handle capitalization better and might perform better on tasks where case sensitivity mat
Domain-Specific BERT Variants:
BioBERT: Pre-trained on biomedical corpora, useful for NER tasks in the medical field.
FinancialBERT: Fine-tuned specifically for financial and business terminology.
Other Pre-trained NER Models: You can also explore models like distilbert-base-cased or roberta-base, which may offer smaller architectu
In Summary:
For General NER: bert-base-uncased should be a great choice. It works well for a wide range of entity recognition tasks and provides a s
For Specialized NER: If you're working with domain-specific entities, it may be worthwhile to explore specialized models or domain-tuned
You can always start with bert-base-uncased and evaluate its performance on your validation set. If the results aren't satisfactory, you







You said:
how to unzip the bert model file
ChatGPT said:
To unzip a BERT model file, typically, the file is in a compressed .tar.gz or .zip format. You can use the following methods based on yo

1. Unzip .zip File:
If your BERT model is in a .zip file format, you can use Python's zipfile module to unzip it.

python
Copy
Edit
import zipfile
import os

# Path to the .zip file
zip_file_path = 'path_to_your_bert_model.zip'
extract_folder = 'path_to_extract_folder'  # Folder to extract files to

# Unzipping the .zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_folder)

print(f"Files extracted to: {extract_folder}")
2. Unzip .tar.gz or .tgz File:
If your model is in a .tar.gz or .tgz format, you can use Python's tarfile module.

python
Copy
Edit
import tarfile
import os

# Path to the .tar.gz file
tar_file_path = 'path_to_your_bert_model.tar.gz'
extract_folder = 'path_to_extract_folder'  # Folder to extract files to

# Unzipping the .tar.gz file
```

```
with tarfile.open(tar_file_path, 'r:gz') as tar_ref:
    tar_ref.extractall(extract_folder)

print(f"Files extracted to: {extract_folder}")
3. Using Command Line (Linux/Mac/Windows with WSL):
You can also unzip the files directly from the command line.

For .zip files:

bash
Copy
Edit
unzip path_to_your_bert_model.zip -d path_to_extract_folder
For .tar.gz files:

bash
Copy
Edit
tar -xvzf path_to_your_bert_model.tar.gz -C path_to_extract_folder
After Unzipping:
Once the model is unzipped, you should see multiple files, typically including:

config.json: Configuration for the model.
pytorch_model.bin: The model weights.
vocab.txt or vocab.json: Tokenizer vocabulary files.
merges.txt (if it's a BPE model like RoBERTa).
You can then load the model using the transformers library or any other BERT-based framework.

python
Copy
Edit
from transformers import BertTokenizer, BertForTokenClassification

# Load the model and tokenizer
model = BertForTokenClassification.from_pretrained(extract_folder)
tokenizer = BertTokenizer.from_pretrained(extract_folder)
This should allow you to successfully unzip and use the BERT model for your task.


To enhance the capability of your NER model by combining manually tagged data with a pre-trained BERT model like bert-base-uncased, here

Step-by-Step Approach:
Step 1: Prepare Manual Tagging Data
You already have a manually tagged dataset, and you want to use this as additional data to fine-tune the BERT model. The key here is to

Step 2: Preprocess and Tokenize the Data
You will need to tokenize the manual tagging data using the BERT tokenizer. This step involves converting each token into its correspond

Step 3: Fine-tune BERT on the Manual Data
Once your manual data is in the correct format, fine-tune the BERT model (bert-base-uncased) on this data. The fine-tuning will adapt th

Step 4: Combine Both BERT and Manual Data
You will combine both the pre-trained BERT model and your manually tagged data by fine-tuning BERT specifically for your use case. This

Here’s how you can implement the process in code:
1. Prepare Your Manually Tagged Data
Let's assume you have manually tagged data in a format like this:

Sentence    Token   BIO Tag
Microsoft acquired LinkedIn for $26 billion.    Microsoft   B-ORG
Microsoft acquired LinkedIn for $26 billion.    acquired    O
Microsoft acquired LinkedIn for $26 billion.    LinkedIn    B-ORG
Microsoft acquired LinkedIn for $26 billion.    for O
Microsoft acquired LinkedIn for $26 billion.    $26 B-MONEY
Microsoft acquired LinkedIn for $26 billion.    billion I-MONEY
Ensure the data is split properly by token and labeled as BIO tags for each token.

2. Preprocessing and Tokenizing with BERT
You will use the transformers library to load the pre-trained BERT model and tokenizer and tokenize your manual data.

python
Copy
Edit
from transformers import BertTokenizer
import pandas as pd

# Initialize the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Assuming your manually tagged data is in a pandas DataFrame
df = pd.read_excel('path_to_manual_tagged_data.xlsx')

# Tokenize each sentence
```

```
def tokenize_and_align_labels(text, labels):
    # Tokenize the text with padding and truncation
    tokenized_input = tokenizer(text.split(), padding='max_length', truncation=True, is_split_into_words=True, return_tensors="pt")

    # Align the BIO labels with the tokenized words
    word_ids = tokenized_input.word_ids()
    labels = [labels[i] if word_ids[i] is not None else 'O' for i in range(len(word_ids))]

    return tokenized_input, labels

# Apply tokenization and BIO alignment to the data
tokenized_data = []
for _, row in df.iterrows():
    sentence = row['Sentence']
    labels = row['BIO Tag'].split()  # Ensure your BIO tags are split into separate words
    tokenized_input, aligned_labels = tokenize_and_align_labels(sentence, labels)
    tokenized_data.append((tokenized_input, aligned_labels))

3. Fine-tune BERT on Your Manual Data
Now, you’ll fine-tune the BERT model using the tokenized data. You will need to prepare a dataset and DataLoader for training.

python
Copy
Edit
from transformers import BertForTokenClassification, Trainer, TrainingArguments
from torch.utils.data import Dataset, DataLoader
import torch

class NERDataset(Dataset):
    def __init__(self, tokenized_data):
        self.tokenized_data = tokenized_data

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

    def __getitem__(self, idx):
        tokenized_input, labels = self.tokenized_data[idx]
        input_ids = tokenized_input['input_ids'].squeeze()  # shape: [max_len]
        attention_mask = tokenized_input['attention_mask'].squeeze()  # shape: [max_len]
        labels = torch.tensor([label2id[label] for label in labels], dtype=torch.long)  # convert labels to ids
        return {'input_ids': input_ids, 'attention_mask': attention_mask, 'labels': labels}

# Initialize model for token classification
model = BertForTokenClassification.from_pretrained('bert-base-uncased', num_labels=len(label2id))

# Create the dataset
dataset = NERDataset(tokenized_data)

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',          # output directory
    evaluation_strategy="epoch",     # evaluate every epoch
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
)

# Initialize the Trainer
trainer = Trainer(
    model=model,                         # the model
    args=training_args,                  # training arguments
    train_dataset=dataset,               # training dataset
)

# Fine-tune the model
trainer.train()

# Save the fine-tuned model
model.save_pretrained('fine_tuned_bert')
4. Evaluating the Model
After fine-tuning, you can evaluate the model using a validation set:

python
Copy
Edit
# After fine-tuning, evaluate the model
trainer.evaluate()
5. Using the Fine-Tuned Model for Inference
Once the model is fine-tuned, you can use it to predict NER tags on new sentences.
```

python
Copy
Edit

```
# Load the fine-tuned model
model = BertForTokenClassification.from_pretrained('fine_tuned_bert')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

def predict_ner(sentence):
    inputs = tokenizer(sentence.split(), return_tensors="pt", is_split_into_words=True)
    outputs = model(**inputs)
    logits = outputs.logits
    predictions = torch.argmax(logits, dim=-1)

    tokens = tokenizer.convert_ids_to_tokens(inputs['input_ids'].squeeze().tolist())
    labels = [id2label[p.item()] for p in predictions.squeeze()]

    return list(zip(tokens, labels))

# Example sentence
sentence = "Microsoft acquired LinkedIn for $26 billion."
predicted_ner = predict_ner(sentence)
print(predicted_ner)
```

6. Hybrid Model Usage: Combining Pre-trained and Manual Data

This approach uses both the power of the pre-trained BERT model (bert-base-uncased) and the manually tagged data to fine-tune the model

By fine-tuning the BERT model on your manually tagged data, you're teaching it to recognize custom entities that may not be part of the

7. Additional Steps:

Create a Validation Set: After fine-tuning, evaluate the model’s performance on a validation set and adjust hyperparameters if needed.
Save the Fine-Tuned Model: You can save the fine-tuned model for later use and deployment.
This approach should meet your client's requirement of combining both pre-trained BERT capabilities and manually tagged data for enhanc

Start coding or generate with AI.

Start coding or generate with AI.