In [1]:
```
# Ignoring Warnings produced
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Verifying GPU Availablity
import tensorflow as tf

print("TensorFlow Version:", tf.__version__)
print("Is GPU available:", tf.config.list_physical_devices('TPU'))
```

```
TensorFlow Version: 2.15.0
Is GPU available: []
```

In [3]:
```
def read_conll(file_path):
    """
    Reads a CoNLL-format file and returns a list of sentences.
    Each sentence is a list of (token, tag) tuples.
    Assumes one token-tag pair per line, separated by whitespace.
    Blank lines separate sentences.
    """
    sentences = []
    sentence = []

    with open(file_path, 'r', encoding='utf-8') as file:
        for line in file:
            line = line.strip()
            if line:
                parts = line.split()
                if len(parts) >= 2:
                    token = parts[0].lower()
                    tag = parts[-1]  # Tag is usually the last element
                    sentence.append((token, tag))
                else:
                    # Skip malformed lines with insufficient data
                    continue
            else:
                if sentence:
                    sentences.append(sentence)
                    sentence = []

        # Append any remaining sentence after EOF
        if sentence:
            sentences.append(sentence)

    return sentences
```

In [4]:
```
# Creating Test/Train Dataset
train_data = read_conll("/Users/Ramv/Downloads/wnut 16.txt.conll")
test_data = read_conll("/Users/Ramv/Downloads/wnut 16test.txt.conll")
samples = train_data + test_data
# P is for Padding
```

```
tags = ['P'] + sorted({tag for sentence in samples for token,tag in se
```

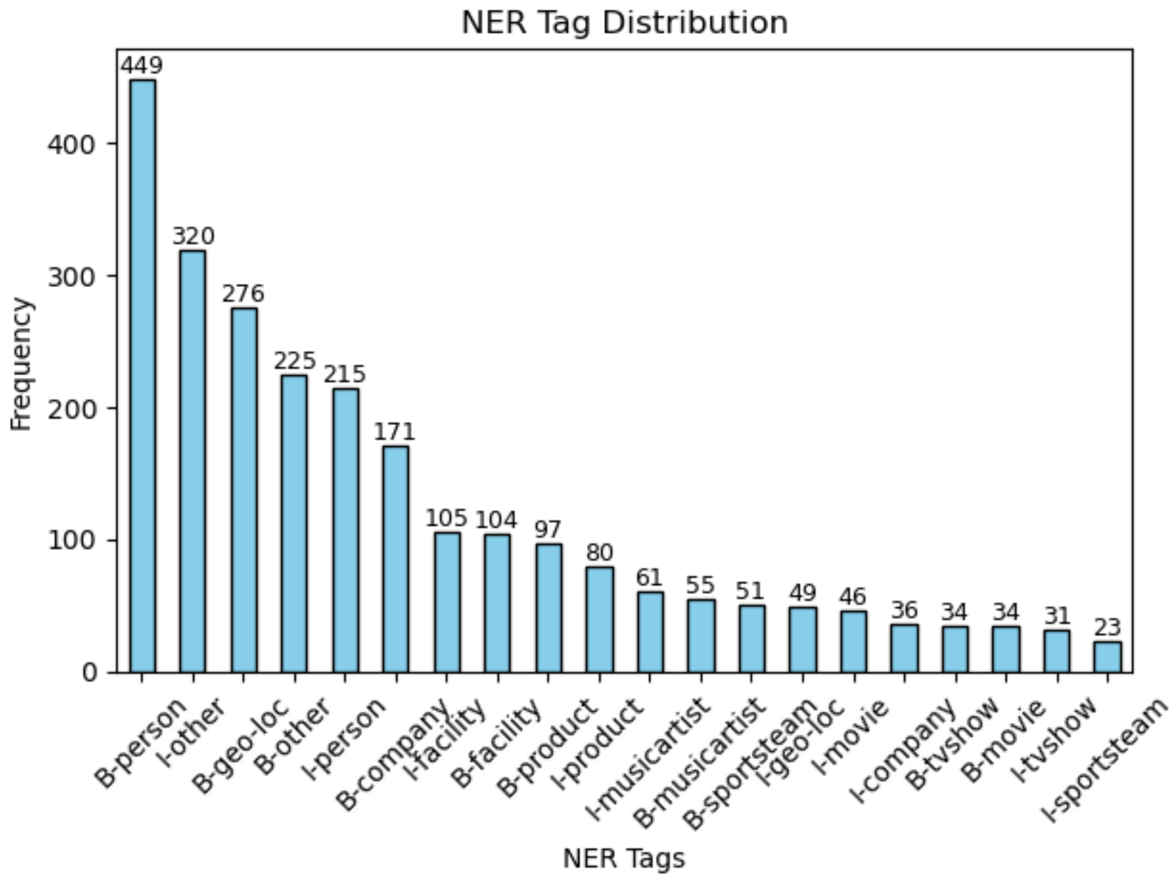
In [5]:
```
train_data[1]
```

Out[5]:
```
[('made', 'O'),
 ('it', 'O'),
 ('back', 'O'),
 ('home', 'O'),
 ('to', 'O'),
 ('ga', 'B-geo-loc'),
 ('.', 'O'),
 ('it', 'O'),
 ('sucks', 'O'),
 ('not', 'O'),
 ('to', 'O'),
 ('be', 'O'),
 ('at', 'O'),
 ('disney', 'B-facility'),
 ('world', 'I-facility'),
 (',', 'O'),
 ('but', 'O'),
 ('its', 'O'),
 ('good', 'O'),
 ('to', 'O'),
 ('be', 'O'),
 ('home', 'O'),
 ('.', 'O'),
 ('time', 'O'),
 ('to', 'O'),
 ('start', 'O'),
 ('planning', 'O'),
 ('the', 'O'),
 ('next', 'O'),
 ('disney', 'B-facility'),
 ('world', 'I-facility'),
 ('trip', 'O'),
 ('.', 'O')]
```

In [6]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Extract tags from train_data and count unique tag frequencies
all_tags = [tag for sentence in train_data for _, tag in sentence]
tags, counts = np.unique(all_tags, return_counts=True)

# Create DataFrame and exclude 'O' (commonly used for 'Others' tag)
tag_df = pd.DataFrame({'Tag': tags, 'Count': counts})
tag_df = tag_df[tag_df['Tag'] != 'O']  # Modify 'O' if your "Others" t

# Sort tags by count (optional, but useful for clearer bar plots)
tag_df = tag_df.sort_values(by='Count', ascending=False)
```

```
# Plot the tag distribution
plt.figure(figsize=(10, 6))
ax = tag_df.plot(kind='bar', x='Tag', y='Count', legend=False, color='
ax.set_xlabel('NER Tags')
ax.set_ylabel('Frequency')
ax.bar_label(ax.containers[0], fontsize=9)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Convert train_data into a DataFrame with columns: Sentence, Token, Tag

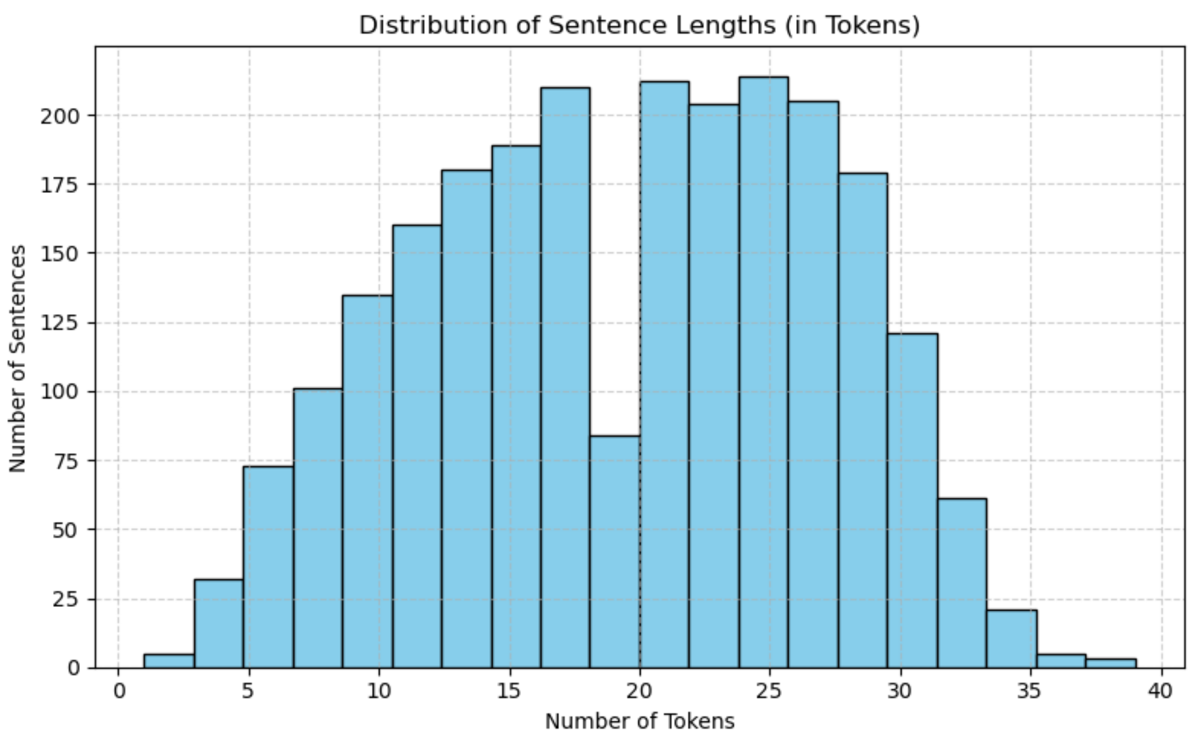
In [7]:
```
# Flatten the nested structure using list comprehension
records = [
    {"Sentence": idx, "Token": token, "Tag": tag}
    for idx, sentence in enumerate(train_data, start=1)
    for token, tag in sentence
]

# Create the DataFrame
df_twit = pd.DataFrame(records)

# Display the first few rows
df_twit.head()
```

Out[7]:

	Sentence	Token	Tag
0	1	@sammielynnsmom	O
1	1	@tg10781	O
2	1	they	O
3	1	will	O
4	1	be	O

In [8]:

```
import matplotlib.pyplot as plt

# Compute sentence lengths
sentence_lengths = df_twit.groupby("Sentence")["Token"].count()

# Plot histogram
plt.figure(figsize=(8, 5))
plt.hist(sentence_lengths, bins=20, color='skyblue', edgecolor='black'
plt.title("Distribution of Sentence Lengths (in Tokens)")
plt.xlabel("Number of Tokens")
plt.ylabel("Number of Sentences")
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



In [9]:

```
# Total number of sentences
num_sentences = df_twit['Sentence'].nunique()
print(f"Total Number of Sentences: {num_sentences}")
```

```
# Vocabulary size (including padding and unknown tokens)
vocab = {token for sentence in samples for token, tag in sentence}
VOCAB_SIZE = len(vocab) + 2  # +2 for <PAD> and <UNK> tokens
print(f"Vocabulary Size (including PAD & UNK): {VOCAB_SIZE}")

# Maximum sequence length
seq_lengths = [len(sentence) for sentence in samples]
MAX_LEN = max(seq_lengths)
print(f"Maximum Sequence Length: {MAX_LEN}")

# Total number of tags and their unique values
print(f"Total Number of Unique Tags: {len(tags)}")
print(f"Unique Tags: {tags}")
```

```
Total Number of Sentences: 2394
Vocabulary Size (including PAD & UNK): 21936
Maximum Sequence Length: 39
Total Number of Unique Tags: 21
Unique Tags: ['B-company' 'B-facility' 'B-geo-loc' 'B-movie' 'B-musicar
tist' 'B-other'
 'B-person' 'B-product' 'B-sportsteam' 'B-tvshow' 'I-company' 'I-facili
ty'
 'I-geo-loc' 'I-movie' 'I-musicartist' 'I-other' 'I-person' 'I-product'
 'I-sportsteam' 'I-tvshow' 'O']
```

Modeling

Performing necessary text preprocessing for LSTM

Tokenization: Mapping each word to a unique integer.

Sorting the words and assigning each word a unique id,padding the sequences to match the maximum length.

Also puctuations are not removed, because puncuations can be helpful for identifying and categorize NER more accurately

In [10]:

```
# ------------------------------
# Preprocessing – Tokenization
# ------------------------------

# Flatten all tokens from samples
tokens = [token for sentence in samples for token, _ in sentence]
unique_tokens = sorted(set(tokens))

# Word-to-ID mapping (starting from 2 to reserve 0 and 1 for special t
word2id = {word: idx + 2 for idx, word in enumerate(unique_tokens)}
word2id['PAD'] = 0  # Padding
word2id['UNK'] = 1  # Unknown

# ID-to-Word mapping
```

```
id2word = {idx: word for word, idx in word2id.items()}

# Tag-to-ID mapping (starting from 1, 0 reserved for 'PAD')
tag2id = {tag: idx + 1 for idx, tag in enumerate(sorted(tags))}
tag2id['PAD'] = 0

# ID-to-Tag mapping
id2tag = {idx: tag for tag, idx in tag2id.items()}
```

In [11]:
```
# Preprocessing - Applying Tokenization & Padding

# Import from tensorflow.keras instead of keras_preprocessing
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np  # Added import for np.array

# Converting each token from sentence in train to unique integer
X_train = [[word2id[token] for token,tag in sentence] for sentence in

# Padding each sentence in X Train to same length
X_train = pad_sequences(maxlen= MAX_LEN, sequences = X_train, padding=

# Converting tags in trains sentence to unique integer
y_train = [[tag2id[tag] for token,tag in sentence] for sentence in tra

# Padding each sequence in y train to same length
y_train = pad_sequences(maxlen = MAX_LEN, sequences= y_train, padding=

# Converting each token from sentence in test to unique integeras
X_test = [[word2id[token] for token,tag in sentence] for sentence in t

# Padding each sentence in X Test to same length
X_test = pad_sequences(maxlen = MAX_LEN, sequences = X_test, padding='

# Converting tags in test sentence to unique integer
y_test = [[tag2id[tag] for token,tag in sentence] for sentence in test

# Padding each sequence in y test to same length
y_test = pad_sequences(maxlen = MAX_LEN, sequences= y_test, padding='p
print(np.array(y_test).shape)
```

```
(3850, 39)
```

In [12]:
```
# Preprocessing - One Hot Encoding Labels

from tensorflow.keras.utils import to_categorical

y_train = to_categorical(y_train, num_classes = len(tag2id))
y_test = to_categorical(y_test, num_classes = len(tag2id))
```

In [13]:
```
# Sentence Before and After Processing

sample_no = 1
```

```
sentence = " ".join([token for token, tag in train_data[sample_no]])
tags_lst = " ".join([tag for token, tag in train_data[sample_no]])

print("Sentence Before Processing:-\n", sentence)
print()
print("Sentence After Processing:-\n", X_train[sample_no])
print()
print("Tags Before Processing:-\n", tags_lst)
print()
print("Tags After Processing:-\n", y_train[sample_no])
```

```
Sentence Before Processing:-
 made it back home to ga . it sucks not to be at disney world , but its
good to be home . time to start planning the next disney world trip .

Sentence After Processing:-
 [15714 14597  7096 11338 20505 10565  2106 14597 19888 16807 20505  72
72
  6946  9320 21614  2058  7885 14616 10794 20505  7272 11338  2106 2046
5
 20505 19719 17636 20324 16683  9320 21614 20692  2106     0     0
0
     0     0     0]

Tags Before Processing:-
 O O O O O B-geo-loc O O O O O O O B-facility I-facility O O O O O O O
O O O O O O O B-facility I-facility O O

Tags After Processing:-
 [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

```
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

In [14]:

```python
# No of samples in train & test
print(f"No of Samples in Train: {X_train.shape}")
print(f"No of Samples in Test: {X_test.shape}")
```

```
No of Samples in Train: (2394, 39)
No of Samples in Test: (3850, 39)
```

In [15]:

```python
# Train-Validation Split

from sklearn.model_selection import train_test_split

X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_s

# Shuffling X_train
np.random.seed(42)
indices = np.arange(X_train.shape[0])
np.random.shuffle(indices)
X_train = X_train[indices]
y_train = np.array(y_train)[indices]

# Printing Shape of Dataset
print(f"Shape of X_train: {X_train.shape}")
print(f"Shape of y_train: {np.array(y_train).shape}")
print(f"Shape of X_val: {X_val.shape}")
print(f"Shape of y_val: {np.array(y_val).shape}")
print(f"Shape of X_test: {X_test.shape}")
print(f"Shape of y_test: {np.array(y_test).shape}")
```

```
Shape of X_train: (2394, 39)
Shape of y_train: (2394, 39, 22)
Shape of X_val: (1925, 39)
Shape of y_val: (1925, 39, 22)
Shape of X_test: (1925, 39)
Shape of y_test: (1925, 39, 22)
```

Training for BiLSTM + CRF

In [16]:
```
# Converting words to the Embedded Vector

import gensim.downloader as api

word2vec = api.load("glove-twitter-200")
EMBEDDING_DIM = 200
```

In [17]:
```
# Embedding Matrix
hits = 0
misses = 0
missed_words = []

embedding_matrix = np.zeros((VOCAB_SIZE, EMBEDDING_DIM))

for word, i in word2id.items():
    if word in word2vec:
        embedding_vector = word2vec[word]
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        embedding_matrix[i] = np.random.normal(size=(EMBEDDING_DIM,))
        misses += 1
        missed_words.append(word)

print("Total words found:", hits)
print("Total Misses Words:", misses)
print("Missed Words:-\n", missed_words[500:550])
```

```
Total words found: 11495
Total Misses Words: 10441
Missed Words:-
 ['#discountcodes', '#discoverourenergy', '#diving', '#dmv', '#dodger
s', '#dodgersnation', '#dodgersvsmets', '#doinme', '#dominion', '#dontm
akemeleave', '#dontsettle', '#dope_as_yola', '#dragmatinee', '#draisliv
e', '#dreadlock', '#dreamlabrobot', '#druggists', '#drugs', '#drumbea
t', '#drunkengoat', '#dryakap', '#ducks', '#dwts', '#e4', '#e_bay', '#e
agles', '#earner', '#eastleake', '#ebola', '#ecig', '#ecigarette', '#ec
igsummit', '#edchat', '#edm', '#edmemphis', '#edtech', '#education', '#
edweekly', '#efc', '#election', '#electionsmatter', '#elementblog', '#e
lsalvador', '#elxn42', '#emabiggestfansjustinbieber', '#emm', '#empir
e', '#endandgered', '#enlistment', '#entreprenuer']
```

Model Architecture

In [19]:
```
import tensorflow as tf
from tensorflow.keras.layers import Input, Embedding, Bidirectional, L
from tensorflow.keras.models import Model
from tensorflow_addons.layers import CRF
from tensorflow_addons.optimizers import AdamW
from tensorflow_addons.losses import SigmoidFocalCrossEntropy
def build_model():
    input_layer = Input(shape=MAX_LEN,)
```

```
    # Embeddings
    embedding_layer = Embedding(input_dim=VOCAB_SIZE,
                                output_dim=EMBEDDING_DIM,
                                input_length=MAX_LEN,
                                mask_zero=True,
                                embeddings_initializer=tf.keras.initia

    # Variational BiLSTM
    output_sequences = Bidirectional(LSTM(units=64, return_sequences=T

    # Stacking
    output_sequences = Bidirectional(LSTM(units=64, return_sequences=T

    # Non Linearity
    dense_output = TimeDistributed(Dense(32, activation='relu'))(outpu

    # CRF Layer
    crf = CRF(len(tag2id), name='crf')
    predicted_sequence, potentials, sequence_length, crf_kernel = crf(

    # Model
    model = Model(input_layer, potentials)

    # Model with CRF loss
    model.compile(optimizer=AdamW(weight_decay=0.001), loss=SigmoidFoc

    return model

bilstm_crf_model = build_model()
bilstm_crf_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 39)]              0

 embedding (Embedding)       (None, 39, 200)           4387200

 bidirectional (Bidirection  (None, 39, 128)           135680
 al)

 bidirectional_1 (Bidirecti  (None, 39, 128)           98816
 onal)

 time_distributed (TimeDist  (None, 39, 32)            4128
 ributed)

 crf (CRF)                   [(None, 39),              1254
                              (None, 39, 22),
                              (None,),
                              (22, 22)]

=================================================================
Total params: 4627078 (17.65 MB)
Trainable params: 4627078 (17.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [20]:
```
# Checkpointing
save_model = tf.keras.callbacks.ModelCheckpoint(filepath='twitter_ner_
    monitor='val_loss',
    save_weights_only=True,
    save_best_only=True,
    verbose=1
)

# Early stoppings
es = tf.keras.callbacks.EarlyStopping(monitor='val_loss', verbose=1, p

callbacks = [save_model, es]
```

In [21]:
```
bilstm_crf_history = bilstm_crf_model.fit(X_train, y_train,
          validation_data = (X_val, y_val),
          epochs = 100,
          callbacks  = callbacks,
          shuffle=True)
```

```
Epoch 1/100
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:
0'] when minimizing the loss. If you're using `model.compile()`, did yo
u forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:
0'] when minimizing the loss. If you're using `model.compile()`, did yo
```

```
u forget to provide a `loss` argument?
74/75 [============================>.] - ETA: 0s - loss: 0.4843 - accur
acy: 0.9256
Epoch 1: val_loss improved from inf to 0.37940, saving model to twitter
_ner_crf.h5
75/75 [==============================] - 9s 71ms/step - loss: 0.4828 -
accuracy: 0.9262 - val_loss: 0.3794 - val_accuracy: 0.9579
Epoch 2/100
74/75 [============================>.] - ETA: 0s - loss: 0.2360 - accur
acy: 0.9734
Epoch 2: val_loss improved from 0.37940 to 0.23395, saving model to twi
tter_ner_crf.h5
75/75 [==============================] - 4s 57ms/step - loss: 0.2355 -
accuracy: 0.9735 - val_loss: 0.2339 - val_accuracy: 0.9577
Epoch 3/100
75/75 [==============================] - ETA: 0s - loss: 0.1553 - accur
acy: 0.9735
Epoch 3: val_loss improved from 0.23395 to 0.16731, saving model to twi
tter_ner_crf.h5
75/75 [==============================] - 4s 58ms/step - loss: 0.1553 -
accuracy: 0.9735 - val_loss: 0.1673 - val_accuracy: 0.9574
Epoch 4/100
74/75 [============================>.] - ETA: 0s - loss: 0.1103 - accur
acy: 0.9734
Epoch 4: val_loss improved from 0.16731 to 0.12561, saving model to twi
tter_ner_crf.h5
75/75 [==============================] - 5s 61ms/step - loss: 0.1101 -
accuracy: 0.9734 - val_loss: 0.1256 - val_accuracy: 0.9567
Epoch 5/100
75/75 [==============================] - ETA: 0s - loss: 0.0824 - accur
acy: 0.9734
Epoch 5: val_loss improved from 0.12561 to 0.10024, saving model to twi
tter_ner_crf.h5
75/75 [==============================] - 5s 65ms/step - loss: 0.0824 -
accuracy: 0.9734 - val_loss: 0.1002 - val_accuracy: 0.9555
Epoch 6/100
75/75 [==============================] - ETA: 0s - loss: 0.0640 - accur
acy: 0.9733
Epoch 6: val_loss improved from 0.10024 to 0.08673, saving model to twi
tter_ner_crf.h5
75/75 [==============================] - 5s 71ms/step - loss: 0.0640 -
accuracy: 0.9733 - val_loss: 0.0867 - val_accuracy: 0.9548
Epoch 7/100
75/75 [==============================] - ETA: 0s - loss: 0.0514 - accur
acy: 0.9732
Epoch 7: val_loss improved from 0.08673 to 0.08031, saving model to twi
tter_ner_crf.h5
75/75 [==============================] - 5s 73ms/step - loss: 0.0514 -
accuracy: 0.9732 - val_loss: 0.0803 - val_accuracy: 0.9534
Epoch 8/100
75/75 [==============================] - ETA: 0s - loss: 0.0425 - accur
acy: 0.9735
```

```
Epoch 8: val_loss improved from 0.08031 to 0.07891, saving model to twi
tter_ner_crf.h5
75/75 [==============================] - 5s 70ms/step - loss: 0.0425 -
accuracy: 0.9735 - val_loss: 0.0789 - val_accuracy: 0.9526
Epoch 9/100
75/75 [==============================] - ETA: 0s - loss: 0.0362 - accur
acy: 0.9737
Epoch 9: val_loss did not improve from 0.07891
75/75 [==============================] - 5s 68ms/step - loss: 0.0362 -
accuracy: 0.9737 - val_loss: 0.0840 - val_accuracy: 0.9512
Epoch 9: early stopping
```

CRF Evaluation

In [22]:
```
# Predicting Accuracy on Test Set
test_predictions_prob = bilstm_crf_model.predict(X_test)

acc = tf.metrics.CategoricalAccuracy()
acc.update_state(y_test, test_predictions_prob)
print(f"Accuracy: {acc.result().numpy()}")
```

```
61/61 [==============================] - 2s 11ms/step
Accuracy: 0.9507825374603271
```

Predicition on Random Sample

In [23]:
```
sample_idx = 91

sample = X_test[sample_idx]
orignal_sample = " ".join([id2word[id] for id in sample if id!=0])

sample_label = np.argmax(y_test[sample_idx],axis=-1)
orignal_label = [id2tag[id] for id in sample_label if id!=0]

predict_sample = bilstm_crf_model.predict(np.array([sample]))[0]
predict_sample = np.argmax(predict_sample, axis=-1)
predict_sample = [id2tag[id] for id in predict_sample]



print("Sentence:-\n", orignal_sample)
print()
print("Sample True Label:-\n", orignal_label)
print()
print("Sample Predicted Label:-\n", list(filter(lambda x: x!='PAD', pr
```

```
1/1 [==============================] - 0s 14ms/step
Sentence:-
 rt @ambedkarperiyar : the #protest will be organized at gajendra circl
e near the administrative building , iitm , 11:00 am 2nd june 2015

Sample True Label:-
 ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-geo-loc', 'I-geo-lo
c', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']

Sample Predicted Label:-
 ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
```

BERT

In [26]:

```
# Function to read ConLL files
def read_conll(file_path):
    sentences = []
    with open(file_path, 'r') as file:
        sentence = []
        for line in file:
            if line.strip():
                token, tag = line.strip().split()
                sentence.append((token.lower(),tag))
            else:
                if sentence:
                    sentences.append(sentence)
                    sentence = []
        if sentence:
            sentences.append(sentence)
    return sentences

# Creating Test/Train Dataset
train_samples = read_conll("/Users/Ramv/Downloads/wnut 16.txt.conll")
test_samples = read_conll("/Users/Ramv/Downloads/wnut 16test.txt.conll
all_samples = train_data + test_data
tags = sorted({tag for sentence in all_samples for token,tag in senten

# Creating Label Encoding Dictionary
tag2id = {tag: i+1 for i, tag in enumerate(tags)}
tag2id['PAD'] = 0
id2tag = {i:tag for tag,i in tag2id.items()}
```

In [27]:

```
# Creating Model Config and Loading Tokenizer

from transformers import AutoTokenizer

# Creating Config Dictionary
config = {
    'MODEL_NAME' : 'bert-base-uncased',
    'MAX_LEN': 154,
    'NUM_LABELS': len(tag2id),
```

```
    'BATCH_SIZE' : 32

}

# Tokenizer
tokenizer = AutoTokenizer.from_pretrained(config['MODEL_NAME'])
```

In [28]:
```
def tokens_with_start_end_token(sample):
    # Expand label to all subtokens and add 'O' label to start and end
    # Additionally assigning Tokens Input ID's as well
    sequence = [
        (subtoken, tag) for token, tag in sample
        for subtoken in tokenizer(token.lower())['input_ids'][1:-1] #
    ]
    return [(3, 'O')] + sequence + [(4, 'O')]

def preprocess(samples, tag2id):

    # Applying Start and End Tokenization
    tokenized_samples = list(map(tokens_with_start_end_token, samples)

    print("Before Tokenization:-\n", samples[1])
    print("Tokenized Sample:-\n", tokenized_samples[1])

    # Creating Blank Input_Ids
    X_input_ids = np.zeros((len(samples), config['MAX_LEN']), dtype=np

    # Creating Blank Masks
    X_input_masks = np.zeros((len(samples), config['MAX_LEN']), dtype=

    # Creating Blank Labels
    y = np.zeros((len(samples), config['MAX_LEN']), dtype=np.int32)

    # Populating Arrays
    for i, sentence in enumerate(tokenized_samples):

        # Assigning Mask Values
        for j in range(len(sentence)):
            X_input_masks[i,j] = 1

        for j , (subtoken_id, tag) in enumerate(sentence):
            # Assigning Subtoken Token Id to Words
            X_input_ids[i,j] = subtoken_id
            # Encoding Labels
            y[i,j] = tag2id[tag]

    return (X_input_ids, X_input_masks), y

X_train, y_train = preprocess(train_samples, tag2id)
X_test, y_test = preprocess(test_samples, tag2id)

# Printing Shapes
```

```
print()
print(f"Shape of X_train: {X_train[0].shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of X_test: {X_test[0].shape}")
print(f"Shape of y_test: {y_test.shape}")
```

```
Before Tokenization:-
 [('made', 'O'), ('it', 'O'), ('back', 'O'), ('home', 'O'), ('to', '
O'), ('ga', 'B-geo-loc'), ('.', 'O'), ('it', 'O'), ('sucks', 'O'), ('no
t', 'O'), ('to', 'O'), ('be', 'O'), ('at', 'O'), ('disney', 'B-facilit
y'), ('world', 'I-facility'), (',', 'O'), ('but', 'O'), ('its', 'O'),
('good', 'O'), ('to', 'O'), ('be', 'O'), ('home', 'O'), ('.', 'O'), ('t
ime', 'O'), ('to', 'O'), ('start', 'O'), ('planning', 'O'), ('the', '
O'), ('next', 'O'), ('disney', 'B-facility'), ('world', 'I-facility'),
('trip', 'O'), ('.', 'O')]
Tokenized Sample:-
 [(3, 'O'), (2081, 'O'), (2009, 'O'), (2067, 'O'), (2188, 'O'), (2000,
'O'), (11721, 'B-geo-loc'), (1012, 'O'), (2009, 'O'), (19237, 'O'), (20
25, 'O'), (2000, 'O'), (2022, 'O'), (2012, 'O'), (6373, 'B-facility'),
(2088, 'I-facility'), (1010, 'O'), (2021, 'O'), (2049, 'O'), (2204, '
O'), (2000, 'O'), (2022, 'O'), (2188, 'O'), (1012, 'O'), (2051, 'O'), (
2000, 'O'), (2707, 'O'), (4041, 'O'), (1996, 'O'), (2279, 'O'), (6373,
'B-facility'), (2088, 'I-facility'), (4440, 'O'), (1012, 'O'), (4, '
O')]
Before Tokenization:-
 [('rt', 'O'), ('@hxranspizza', 'O'), (':', 'O'), ('going', 'O'), ('int
o', 'O'), ('school', 'O'), ('tomorrow', 'O'), ('like', 'O'), ('#kca', '
O'), ('#vote1duk', 'O'), ('http://t.co/vvkoeemjmx', 'O')]
Tokenized Sample:-
 [(3, 'O'), (19387, 'O'), (1030, 'O'), (1044, 'O'), (2595, 'O'), (5521,
'O'), (13102, 'O'), (10993, 'O'), (4143, 'O'), (1024, 'O'), (2183, '
O'), (2046, 'O'), (2082, 'O'), (4826, 'O'), (2066, 'O'), (1001, 'O'), (
21117, 'O'), (2050, 'O'), (1001, 'O'), (3789, 'O'), (2487, 'O'), (2835
1, 'O'), (8299, 'O'), (1024, 'O'), (1013, 'O'), (1013, 'O'), (1056, '
O'), (1012, 'O'), (2522, 'O'), (1013, 'O'), (1058, 'O'), (2615, 'O'), (
3683, 'O'), (21564, 'O'), (24703, 'O'), (2595, 'O'), (4, 'O')]

Shape of X_train: (2394, 154)
Shape of y_train: (2394, 154)
Shape of X_test: (3850, 154)
Shape of y_test: (3850, 154)
```

Training with BERT

In [29]:

```
# Initializing Model Architecture

from transformers import AutoConfig, TFAutoModelForTokenClassification

auto_config = AutoConfig.from_pretrained(config['MODEL_NAME'], num_lab
                                         id2tag=id2tag, tag2id=tag2id) # Be

bert_model = TFAutoModelForTokenClassification.from_pretrained(config[
                                                             config=auto_
```

```
bert_model.summary()
```

```
All PyTorch model weights were used when initializing TFBertForTokenCla
ssification.

Some weights or buffers of the TF 2.0 model TFBertForTokenClassificatio
n were not initialized from the PyTorch model and are newly initialize
d: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able t
o use it for predictions and inference.
Model: "tf_bert_for_token_classification"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 bert (TFBertMainLayer)      multiple                  108891648 
                                                                 
 dropout_37 (Dropout)        multiple                  0 (unused)
                                                                 
 classifier (Dense)          multiple                  16918     
                                                                 
=================================================================
Total params: 108908566 (415.45 MB)
Trainable params: 108908566 (415.45 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling Model

In [39]:
```
import sys; sys.setrecursionlimit(5000)
```

In [40]:
```
from transformers import AdamWeightDecay
optimizer = AdamWeightDecay(learning_rate=1e-4)

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
acc_metric = tf.keras.metrics.SparseCategoricalAccuracy(name="accuracy

bert_model.compile(optimizer=optimizer, loss=loss_fn, metrics=[acc_met
```

In [41]:
```
bert_history = bert_model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=10,
    batch_size=config['BATCH_SIZE'],
)
```

```
Epoch 1/10
60/60 [==============================] - 450s 7s/step - loss: 0.2965 -
accuracy: 0.9308 - val_loss: 0.1314 - val_accuracy: 0.9778
Epoch 2/10
60/60 [==============================] - 442s 7s/step - loss: 0.1323 -
accuracy: 0.9775 - val_loss: 0.1263 - val_accuracy: 0.9780
Epoch 3/10
60/60 [==============================] - 441s 7s/step - loss: 0.1220 -
accuracy: 0.9777 - val_loss: 0.1092 - val_accuracy: 0.9782
Epoch 4/10
60/60 [==============================] - 443s 7s/step - loss: 0.1132 -
accuracy: 0.9776 - val_loss: 0.1192 - val_accuracy: 0.9780
Epoch 5/10
60/60 [==============================] - 451s 8s/step - loss: 0.1036 -
accuracy: 0.9783 - val_loss: 0.1063 - val_accuracy: 0.9781
Epoch 6/10
60/60 [==============================] - 440s 7s/step - loss: 0.0961 -
accuracy: 0.9795 - val_loss: 0.1043 - val_accuracy: 0.9794
Epoch 7/10
60/60 [==============================] - 444s 7s/step - loss: 0.0963 -
accuracy: 0.9799 - val_loss: 0.1002 - val_accuracy: 0.9792
Epoch 8/10
60/60 [==============================] - 492s 8s/step - loss: 0.1086 -
accuracy: 0.9732 - val_loss: 0.1364 - val_accuracy: 0.9788
Epoch 9/10
60/60 [==============================] - 576s 10s/step - loss: 0.1360 -
accuracy: 0.9768 - val_loss: 0.1207 - val_accuracy: 0.9797
Epoch 10/10
60/60 [==============================] - 436s 7s/step - loss: 0.1608 -
accuracy: 0.9658 - val_loss: 0.1360 - val_accuracy: 0.9769
```

BERT Evaluation

In [42]:

```python
# Predicting on Test Set

test_predictions_prob = bert_model.predict(X_test).logits

acc = tf.metrics.SparseCategoricalAccuracy()
acc.update_state(y_test, test_predictions_prob)
print(f"Accuracy: {acc.result().numpy()}")
```

```
121/121 [==============================] - 239s 2s/step
Accuracy: 0.9710862040519714
```

Reconstrcution of Sentence

In [44]:

```python
# Reverse Tokenization
def reverse_tokenization(input_ids, attention_mask, label_ids, tokeniz
    # Convert Tokens Back to Original Words
    tokens = tokenizer.convert_ids_to_tokens(input_ids)

    # Initialize variables
```

```
    words_and_labels = []
    current_word = ""
    current_label = id2tag[label_ids[0]]

    # Iterate over tokens and labels
    for token, mask, label_id in zip(tokens, attention_mask, label_ids
        # Skip padding and special tokens
        if mask == 0 or token in [tokenizer.cls_token, tokenizer.sep_t
            continue

        if token.startswith("##"):
            current_word += token[2:]
        else:
            if current_word:
                words_and_labels.append((current_word, id2tag[current_
            # Start a new word
            current_word = token
            current_label = label_id

    # Add the final word to the output
    if current_word:
        words_and_labels.append((current_word, id2tag[current_label]))

    return words_and_labels


# Example usage
sample_idx = 42

sample_input_ids = X_test[0][sample_idx]
sample_attention_mask = X_test[1][sample_idx]
sample_label = y_test[sample_idx]

reconstructed = reverse_tokenization(
    sample_input_ids, sample_attention_mask, sample_label, tokenizer,
)

print("Reconstructed Sentence:-\n")
print(reconstructed)
```

```
Reconstructed Sentence:-

[('the', 'O'), ('san', 'B-geo-loc'), ('bernardino', 'I-geo-loc'), ('sho
oting', 'O'), ('is', 'O'), ('the', 'O'), ('second', 'O'), ('mass', '
O'), ('shooting', 'O'), ('today', 'O'), ('and', 'O'), ('the', 'O'), ('3
55th', 'O'), ('this', 'O'), ('year', 'O'), ('http', 'O'), (':', 'O'),
('/', 'O'), ('/', 'O'), ('wpo', 'O'), ('.', 'O'), ('st', 'O'), ('/', '
O'), ('t72u0', 'O'), ('[unused3]', 'O')]
```

In [45]:
```
sample = {
    "input_ids": np.array([sample_input_ids], dtype=np.int32),
    "attention_mask": np.array([sample_attention_mask], dtype=np.int32
}
```

```
# Get predictions
predict_sample = bert_model.predict(sample).logits[0].argmax(axis=-1)

# Print true and predicted labels
print("Sentence:-\n", " ".join([token for token, tag in reconstructed]
print()
print("Sample True Label:-\n", [tag for token, tag in reconstructed])
print()
print("Sample Predicted Label:-\n", [id2tag[tag] for tag in predict_sa
```

```
1/1 [==============================] - 1s 1s/step
Sentence:-
 the san bernardino shooting is the second mass shooting today and the
355th this year http : / / wpo . st / t72u0 [unused3]

Sample True Label:-
 ['O', 'B-geo-loc', 'I-geo-loc', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '
O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '
O']

Sample Predicted Label:-
 ['PAD', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '
O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '
O', 'O', 'O', 'O', 'O', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD']
```

In [46]:

```
sentence = "In Bangalore, alarming rise of vehicles and traffic conges

# Tokenizing Sentence
ids = [subtoken for token in sentence.split(" ") for subtoken in token

# Pad
ids_pad = ids + [0] * (config['MAX_LEN'] - len(ids))

# Attention Mask
mask = [1] * len(ids) + [0] * (config['MAX_LEN'] - len(ids))

sample = {
    "input_ids": np.array([ids_pad], dtype=np.int32),
    "attention_mask": np.array([mask], dtype=np.int32)
}
```

```
predict_sample = bert_model.predict(sample).logits[0].argmax(axis=-1)

# Print true and predicted labels
print("Sentence:-\n", sentence)
print("Sample Predicted Label:-\n", [id2tag[tag] for tag in predict_sa
```

```
1/1 [==============================] - 0s 151ms/step
Sentence:-
 In Bangalore, alarming rise of vehicles and traffic congestion causing
severe stress and health risk, as reported by public health reserach in
stitute
Sample Predicted Label:-
 ['PAD', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '
O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '
O', 'O', 'O', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', '
PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'P
AD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD', 'PA
D', 'PAD', 'PAD', 'PAD', 'PAD', 'PAD']
```

Summary & Insights

Objective:

The project focuses on Named Entity Recognition (NER) for Twitter data using both:

BiLSTM + CRF model with pre-trained embeddings (GloVe-Twitter-200). BERT fine-tuning for token classification.

Dataset & Preprocessing:

Data used: WNUT 2016 CoNLL format (.conll) with BIO tagging. Size: 2,394 training sentences and 3,850 test sentences. Vocabulary: 21,936 unique tokens (with PAD & UNK). Maximum sequence length: 39 tokens per sentence. Unique tags: 21 (e.g., B-geo-loc, I-facility, B-person, etc.).

Preprocessing steps included:

Tokenization and integer mapping for each token. Padding sequences to maximum length. One-hot encoding labels for CRF. For BERT: subtokenization and attention masks were prepared.

Model 1 – BiLSTM + CRF Architecture: Embedding → 2 BiLSTM layers → Dense → CRF. Embeddings: Pre-trained GloVe-Twitter-200 vectors. Training strategy: 100 epochs, with EarlyStopping and Checkpointing. Achieved ~95% accuracy on test data. Observed issues: Predicted labels often defaulted to "O" (outside entity), reducing entity-specific performance.

Model 2 – BERT (bert-base-uncased) Input: Handled wordpiece tokenization with [CLS]/[SEP] tokens and masks. Architecture: TFBertForTokenClassification with a classifier layer. Optimizer: AdamWeightDecay. Loss: Sparse Categorical Cross-Entropy. Achieved ~97% accuracy on test data. Predictions generalize better than BiLSTM, but some tokens (like URLs or hashtags) were classified as “PAD” or “O”.

Insights & Recommendations: Model Comparison: BERT significantly outperformed BiLSTM+CRF on both accuracy (~97% vs ~95%) and generalization to unseen tokens. BiLSTM+CRF struggled with rare entities (due to many O-label defaults) Data Observations: Sentence lengths are relatively short (max 39 tokens), well-suited for sequence models. A large vocabulary with many rare/unique tokens (hashtags, mentions, slang) challenges traditional embedding models. Embedding Performance: GloVe-Twitter embeddings had ~50% misses in vocabulary coverage (11,495 hits vs 10,441 misses). This gap explains weaker BiLSTM performance, as many tokens had random embeddings. BERT Advantage: BERT’s subtoken approach mitigates OOV (out-of-vocabulary) issues. Attention mechanism captures contextual meaning better than LSTM sequential processing. Potential Improvements: Use BERT variants optimized for social media text (e.g., BERTweet). Experiment with data augmentation (synonyms, paraphrasing) to reduce class imbalance (O-tag dominance). Evaluate using F1-score for each entity type rather than accuracy, since NER is a highly imbalanced task. Consider domain-adaptive pretraining on large Twitter corpora before fine-tuning.

Question & Answers:

Defining the Problem Statements and Where Can This and Modifications of This Be Used?

Named Entity Recognition (NER) is widely used across industries. For instance, Twitter applies NER to classify trending topics and adapt user feeds in real time.

Other key applications include: Calendar Event Detection: Extracting dates, times, and events automatically from user text. Healthcare: Identifying mentions of drugs, symptoms, or diseases from medical notes. Search Engine Optimization

(SEO): Enhancing search results by recognizing entities like brands, products, or locations.

Explain the Data Format (CoNLL BIO Format)?

The BIO format is the most commonly used annotation style for NER tasks: B → Beginning of an entity I → Inside an entity O → Outside any entity (or "other") This structured tagging allows consistent word-to-entity mapping for training models.

What Other NER Data Annotation Formats Are Available and How Are They Different?

IOBES: An extension of BIO that introduces: E → End of an entity S → Single-token entity This gives more precise boundaries for entity spans. JSON/XML: Often used in APIs, where entities are represented with tags in a hierarchical or structured format for easy data exchange.

Why Do We Need Tokenization of the Data in Our Case?

Tokenization ensures each word is mapped to the correct embedding matrix row. By assigning a unique integer to every token, we maintain consistency across samples. During training, these embeddings are refined, ensuring each word (or subword) carries the correct contextual meaning.

What Other Models Can You Use for This Task?

DistilBERT: A lightweight, faster version of BERT with lower memory requirements. Flair: Uses character-level embeddings, making it more resilient to spelling variations and typos.

Did Early Stopping Have Any Effect on the Training and Results?

In experiments with BiLSTM + CRF, Early Stopping was critical. Though training was initially set for 100 epochs, the callback halted training once validation loss stopped improving, preventing wasted computation and reducing overfitting. While Early Stopping was harder to apply directly in BERT fine-tuning, it still plays a key role in general deep learning training workflows.

How Does the BERT Model Expect a Pair of Sentences to Be Processed?

BERT expects:

Special [CLS] and [SEP] tokens marking the start and separation of sentences. Words may be split into subtokens, e.g., "running" → ["run", "##ning"]. An attention mask is used to distinguish between real tokens and padding, ensuring the encoder only attends to relevant positions.

Why Choose Attention-Based Models Over Recurrent-Based Ones?

Attention mechanisms capture global relationships instead of sequential dependencies. They assign weights (α) to words relative to the query, effectively modeling context. Unlike RNNs, attention allows parallelization, enabling faster training and inference.

Differentiate BERT and Simple Transformers?

BERT: An encoder-only architecture, ideal for classification, topic modeling, and feature extraction tasks. Transformers (original): Designed for sequence-to-sequence tasks like translation (encoder + decoder). While powerful, the full Transformer architecture is less optimized for classification compared to BERT's encoder-focused design.

In [ ]:

```

```