End-To-End Named Entity Recognition (NER) with Ray and PyTorch

This notebook takes you through an end-to-end NER use-case using Ray's distributed processing, training, and serving capabilities. This notebook utilizes the BioNER dataset. By the end of this notebook, you will have:

- Performed standard NER data processing steps like tokenization, lemmatization using Ray Core and Ray Data
- Fine-tuned the base distilbert using Ray Train
- Gained an understanding how to use Ray Train to configure your model training, evaluation metrics, checkpointing, etc.
- Utilized Ray Serve to create an endpoint for your model that offers fast inference

In [1]:
```
import ray
import ray.data
```

In [2]:
```
# disable logging
import logging
logging.getLogger().disabled = True
```

In [ ]:
```
ray.init()
```

In [4]:
```
# View resources available to Ray - this should match your machine's hardwar
ray.cluster_resources()
```

Out[4]:
```
{'CPU': 8.0,
 'node:__internal_head__': 1.0,
 'node:127.0.0.1': 1.0,
 'object_store_memory': 801708441.0,
 'memory': 1603416884.0}
```

Load BioNER datasets

In [5]:
```
import glob
file_paths = glob.glob("./NERData/**/train.tsv", recursive=True)
file_paths
```

Out[5]:
```
['./NERData\\BC2GM\\train.tsv',
 './NERData\\BC4CHEMD\\train.tsv',
 './NERData\\BC5CDR-chem\\train.tsv',
 './NERData\\BC5CDR-disease\\train.tsv',
 './NERData\\JNLPBA\\train.tsv',
 './NERData\\linnaeus\\train.tsv',
 './NERData\\NCBI-disease\\train.tsv',
 './NERData\\s800\\train.tsv']
```

In [6]:
```python
# define helper functions to read tsv file - sourced from https://github.com
def _read_tsv_data(input_file, fetch_limit = 100):
    """Reads a BIO data. Use fetch_limit to limit the number of l"""
    inpFilept = open(input_file)
    lines = []
    words = []
    labels = []

    counter = 0
    for lineIdx, line in enumerate(inpFilept):
        contents = line.splitlines()[0]
        lineList = contents.split()
        if len(lineList) == 0: # For blank line
            if counter > fetch_limit - 1:
                break
            assert len(words) == len(labels), "lineIdx: %s,  len(words)(%s)
            if len(words) != 0:
                wordSent = " ".join(words)
                labelSent = " ".join(labels)
                lines.append((labelSent, wordSent))
                words = []
                labels = []
                counter += 1
            else:
                print("Two continual empty lines detected!")
        else:
            words.append(lineList[0])
            labels.append(lineList[-1])

    if len(words) != 0 and counter < (fetch_limit - 1):
        wordSent = " ".join(words)
        labelSent = " ".join(labels)
        lines.append((labelSent, wordSent))
        words = []
        labels = []

    inpFilept.close()
    return lines

# Wrapping this function as a ray task for experimentation
@ray.remote
def _read_tsv_data_remote(input_file, fetch_limit):
    return _read_tsv_data(input_file, fetch_limit)
```

The below cell contains a sample of what the loaded dataset looks like. BioNER is a collection of 8 smaller datasets (each covering different topics), denoted by *the 8 different folders - each has a train.tsv file, which contains a collection of sentences and NER tags. The three NER tags are:*

- B (Beginning): Indicates the first token of a named entity (biology related entities, in this case).
- I (Inside): Marks subsequent tokens inside the same named entity.
- O (Outside): Denotes tokens that do not belong to any named entity

The helper function above converts these files into arrays of tuples, where tuple[0] = NER tags, and tuple[1] = sentence corresponding to those tags

In [7]:
```
# Modify this to increase/decrease size of all datasets used downstream
fetch_limit = 5
```

In [8]:
```
# sample output
sample_output = _read_tsv_data(file_paths[0], fetch_limit)
print(sample_output[:3])
len(_read_tsv_data(file_paths[0], fetch_limit))
```

```
[('O O O O O B I I O O O O O O O O B I I O O O O O O O O O O O O B O O O O O
O O O O O O O O O O O', 'Immunohistochemical staining was positive for S - 1
00 in all 9 cases stained , positive for HMB - 45 in 9 ( 90 % ) of 10 , and
negative for cytokeratin in all 9 cases in which myxoid melanoma remained in
the block after previous sections .'), ('B I O O O O O B O O O O O O O B I O
O O B I I O O O O O O O O O O O O O O O O B I O', 'Chloramphenicol acetyltra
nsferase assays examining the ability of IE86 to repress activity from the H
CMV major IE promoter or activate the HCMV early promoter for the 2 . 2 - kb
class of RNAs demonstrated the functional integrity of the IE86 protein .'),
('O O B I I O O O O O O B O O O O O', 'A new DNA repair gene from Schizosacc
haromyces pombe with homology to RecA was identified and characterized .')]
```

Out[8]: 5

In [9]:
```
%%time
# Native Python version - this is a single threaded, sequential way of readi
for file in file_paths:
    _read_tsv_data(file, fetch_limit)
```

```
CPU times: total: 46.9 ms
Wall time: 34.7 ms
```

In [10]:
```
%%time
# Ray task version - notice how CPU time is much lower in this case. This is
futures = [_read_tsv_data_remote.remote(file, fetch_limit) for file in file_
_ = ray.get(futures)
```

```
CPU times: total: 531 ms
Wall time: 1.8 s
```

In [11]:
```
%%time
# Using Ray data to provide a lazy-evaluatable interface to our training dat
from typing import Any, Dict
def parse_file(row: Dict[str, Any]) -> Dict[str, Any]:
    return {"parsed_file": _read_tsv_data(row['file_path'], fetch_limit)} #
```

```
ray_ds = ray.data.from_items([{"file_path": path} for path in file_paths])
processed_ds = ray_ds.map(parse_file)
```

```
CPU times: total: 688 ms
Wall time: 13.5 s
```

(Depends on fetch_limit - for small sizes, native Python is quicker as Ray adds overhead) As seen above, the Ray task version performs best, as it essentially reads the input training files in parallel. For the purposes of this exercise, however, we will be going with the Ray data version as it more closely resembles what one would do for a larger/production scale dataset

Lemmatize and Tokenize Data using Ray

After we've loaded data, the next step is to to perform some processing on it to make it more useful to the model. Specifically, we will be performing:

- Lower-casing the sentences
- Lemmatization, i.e., converting words to their root form. For example, cats would be converted to cat. This helps remove distractions and improves language model understanding
- Converting sentences to inputs the LLM will understand (i.e., tokens) using the appropriate tokenizer model. We also pad the sentences to the max length accepted by the model to allow us to use multiple batches per forward pass in our training loop

In [12]:

```
import nltk
nltk.download('punkt_tab', download_dir='C:\\Users\\Varun Jadia\\Desktop\\co
nltk.download('wordnet', download_dir='C:\\Users\\Varun Jadia\\Desktop\\codi
```

```
[nltk_data] Downloading package punkt_tab to C:\Users\Varun
[nltk_data]     Jadia\Desktop\coding_assignments\ray\ray_venv\nltk_dat
[nltk_data]     a...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\Varun
[nltk_data]     Jadia\Desktop\coding_assignments\ray\ray_venv\nltk_dat
[nltk_data]     a...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[12]: True

In [13]:

```
# Example of nltk lemmatizer
from nltk.stem import WordNetLemmatizer

sentence = "The cats are sitting on the bed."
words = nltk.word_tokenize(sentence)
lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
print(lemmatized_words)
```

```
['The', 'cat', 'are', 'sitting', 'on', 'the', 'bed', '.']
```

In [14]:
```python
def lemmatize_tokenize_and_align_labels(batch):
    from transformers import AutoTokenizer
    import torch

    # Simple dict to map NER tags to categorical variables
    label_to_int = {
        'B': 0,
        'I': 1,
        'O': 2
    }

    tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-v1.1")
    max_length = 512

    parsed_files = batch['parsed_file']
    tokenized_inputs = []

    for parsed_file in parsed_files:
        for i, (label_str, sentence) in enumerate(parsed_file):
            label_list = label_str.split()
            words = sentence.lower().split()  # Convert to lower case

            if len(label_list) != len(words):
                raise ValueError(f"Mismatch: {len(label_list)} labels but {l

            input_ids = []
            aligned_labels = []

            for word, label in zip(words, label_list):
                lemmatized_word = lemmatizer.lemmatize(word)
                word_tokens = tokenizer.tokenize(lemmatized_word)

                if not word_tokens:
                    continue

                word_ids = tokenizer.convert_tokens_to_ids(word_tokens)
                input_ids.extend(word_ids)
                aligned_labels.append(label_to_int[label])

                if len(input_ids) > max_length - 2:
                    break # early break if > seq length

                if len(word_tokens) > 1:
                    if label == 'B':
                        remaining_label = 'I'
                    else:
                        remaining_label = label  # Either 'I' or 'O'

                    for _ in range(len(word_tokens) - 1):
                        aligned_labels.append(label_to_int[remaining_label])

            # Truncate if longer than max_length (accounting for special tok
            if len(input_ids) > max_length - 2:  # -2 for [CLS] and [SEP]
                input_ids = input_ids[:max_length - 2]
                aligned_labels = aligned_labels[:max_length - 2]
```

```python
            final_input_ids = [tokenizer.cls_token_id] + input_ids + [tokeni
            final_labels = [-100] + aligned_labels + [-100]
            attention_mask = [1] * len(final_input_ids)

            # Padding any sentences smaller than max_length
            padding_length = max_length - len(final_input_ids)
            if padding_length > 0:
                final_input_ids += [tokenizer.pad_token_id] * padding_length
                attention_mask += [0] * padding_length
                final_labels += [-100] * padding_length

            tokenized_inputs.append({
                'input_ids': final_input_ids,
                'attention_mask': attention_mask,
                'labels': final_labels
            })

    return {"tokenized_inputs": tokenized_inputs}
```

In [15]:
```python
tokenized_ds = processed_ds.map_batches(lemmatize_tokenize_and_align_labels,
tokenized_ds.count()
```

```
2025-03-01 19:28:48,660 INFO streaming_executor.py:108 -- Starting execution of Dataset. Full logs are in C:\Users\VARUNJ~1\AppData\Local\Temp\ray\session_2025-03-01_19-28-07_605513_15472\logs\ray-data
2025-03-01 19:28:48,662 INFO streaming_executor.py:109 -- Execution plan of Dataset: InputDataBuffer[Input] -> TaskPoolMapOperator[Map(parse_file)->MapBatches(lemmatize_tokenize_and_align_labels)] -> AggregateNumRows[AggregateNumRows]
Running 0: 0.00 row [00:00, ? row/s]
- Map(parse_file)->MapBatches(lemmatize_tokenize_and_align_labels) 1: 0.00 row [00:00, ? row/s]
- AggregateNumRows 2: 0.00 row [00:00, ? row/s]
```

Out[15]: 40

Note that the Ray Data only materializes/evaluates the data when requested, as in the below cell using take_batch (there are other APIs that allow for accessing data from a Ray Data object, like to_pandas(), etc.). Also note how you can chain transformations from one data object to another using map_batches

In [16]:
```python
# inspect tokenized_ds
check_batches = tokenized_ds.take_batch(batch_size=2)
len(check_batches['tokenized_inputs'])
```

```
2025-03-01 19:29:14,247 INFO streaming_executor.py:108 -- Starting execution of Dataset. Full logs are in C:\Users\VARUNJ~1\AppData\Local\Temp\ray\session_2025-03-01_19-28-07_605513_15472\logs\ray-data
2025-03-01 19:29:14,252 INFO streaming_executor.py:109 -- Execution plan of Dataset: InputDataBuffer[Input] -> TaskPoolMapOperator[Map(parse_file)->MapBatches(lemmatize_tokenize_and_align_labels)] -> LimitOperator[limit=2]
Running 0: 0.00 row [00:00, ? row/s]
- Map(parse_file)->MapBatches(lemmatize_tokenize_and_align_labels) 1: 0.00 row [00:00, ? row/s]
```

```
- limit=2 2: 0.00 row [00:00, ? row/s]
```

Out[16]:
```
2
```

Finetune DistilBERT on BioNER dataset

The goal of this section is to finetune the DistilBERT model (a smaller and faster version of the canonical BERT model) on the BioNER dataset to improve performance. We'll analyze the base model's performance (which we expect to be bad) first before running our fine-tuning loop. For this exercise, we will only be fine-tuning the classification head of the model to make our weight updates quicker

In [17]:
```python
# Ray data connects directly to torch dataloader
# NOTE: Ensure using torch 2.3.0 to ensure libuv backend is not used
import torch
from torch.utils.data import Dataset, DataLoader

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

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

    def __getitem__(self, idx):
        item = self.data[idx]
        return {
            'input_ids': torch.tensor(item['input_ids'], dtype=torch.int64),
            'attention_mask': torch.tensor(item['attention_mask'], dtype=tor
            'labels': torch.tensor(item['labels'], dtype=torch.int64)
        }
```

In [18]:
```python
dl = DataLoader(TokenizedDataset(check_batches['tokenized_inputs']), batch_s
for data in dl:
    print(data)
    break
```

```
{'input_ids': tensor([[  101, 13280, 13601,  2728, 27516,  2430, 16710, 2475
4,  1158, 20049,
          3112,  1111,   188,   118,  1620,  1107,  1155,   130,  1692,  972
9,
           117,  3112,  1111,   177, 12913,   118,  2532,  1107,   130,   11
3,
          3078,   110,   114,  1104,  1275,   117,  1105,  4366,  1111,   17
2,
         25669, 26218, 11745,  1179,  1107,  1155,   130,  1692,  1107,  113
4,
          1139,  1775,  7874,  1143,  4371,  7903,  1915,  1107,  1103,  351
0,
          1170,  2166,  2237,   119,   102,     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,     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,     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,     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,     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,
             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,
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,     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,     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,     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,     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,     0,     0,     0,     0,     0,
0,
              0,     0]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0]]), 'labels': tensor([[-100,    2,    2,
2,    2,    2,    2,    2,    2,    2,    2,    2,
            0,    1,    1,    2,    2,    2,    2,    2,    2,    2,    2,
0,
            1,    1,    1,    2,    2,    2,    2,    2,    2,    2,    2,
2,
            2,    2,    2,    0,    1,    1,    1,    1,    2,    2,    2,
2,
            2,    2,    2,    2,    2,    2,    2,    2,    2,    2,    2,
2,
            2,    2,    2,    2, -100, -100, -100, -100, -100, -100, -100, -
100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
```

```
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
          -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
```

```
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
        -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -
100,
        -100, -100, -100, -100, -100, -100, -100, -100]])}
```

```
A module that was compiled using NumPy 1.x cannot be run in
NumPy 2.2.2 as it may crash. To support both 1.x and 2.x
versions of NumPy, modules must be compiled with NumPy 2.0.
Some module may need to rebuild instead e.g. with 'pybind11>=2.12'.

If you are a user of the module, the easiest solution will be to
downgrade to 'numpy<2' or try to upgrade the affected module.
We expect that some modules will need time to support NumPy 2.

Traceback (most recent call last):  File "<frozen runpy>", line 198, in _run
_module_as_main
  File "<frozen runpy>", line 88, in _run_code
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\ipykernel_launcher.py", line 18, in <module>
    app.launch_new_instance()
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\traitlets\config\application.py", line 1075, in launch_instance
    app.start()
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\ipykernel\kernelapp.py", line 739, in start
    self.io_loop.start()
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\tornado\platform\asyncio.py", line 205, in start
    self.asyncio_loop.run_forever()
  File "C:\Users\Varun Jadia\AppData\Local\Programs\Python\Python312\Lib\asy
ncio\base_events.py", line 640, in run_forever
    self._run_once()
  File "C:\Users\Varun Jadia\AppData\Local\Programs\Python\Python312\Lib\asy
ncio\base_events.py", line 1992, in _run_once
    handle._run()
  File "C:\Users\Varun Jadia\AppData\Local\Programs\Python\Python312\Lib\asy
ncio\events.py", line 88, in _run
    self._context.run(self._callback, *self._args)
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\ipykernel\kernelbase.py", line 545, in dispatch_queue
    await self.process_one()
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\ipykernel\kernelbase.py", line 534, in process_one
    await dispatch(*args)
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\ipykernel\kernelbase.py", line 437, in dispatch_shell
    await result
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\ipykernel\ipkernel.py", line 362, in execute_request
    await super().execute_request(stream, ident, parent)
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\ipykernel\kernelbase.py", line 778, in execute_request
    reply_content = await reply_content
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\ipykernel\ipkernel.py", line 449, in do_execute
    res = shell.run_cell(
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\ipykernel\zmqshell.py", line 549, in run_cell
    return super().run_cell(*args, **kwargs)
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\IPython\core\interactiveshell.py", line 3075, in run_cell
```

```
    result = self._run_cell(
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\IPython\core\interactiveshell.py", line 3130, in _run_cell
    result = runner(coro)
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\IPython\core\async_helpers.py", line 128, in _pseudo_sync_runner
    coro.send(None)
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\IPython\core\interactiveshell.py", line 3334, in run_cell_async
    has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\IPython\core\interactiveshell.py", line 3517, in run_ast_nodes
    if await self.run_code(code, result, async_=asy):
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\IPython\core\interactiveshell.py", line 3577, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "C:\Users\Varun Jadia\AppData\Local\Temp\ipykernel_15472\2198905816.p
y", line 2, in <module>
    for data in dl:
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\torch\utils\data\dataloader.py", line 631, in __next__
    data = self._next_data()
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\torch\utils\data\dataloader.py", line 675, in _next_data
    data = self._dataset_fetcher.fetch(index)  # may raise StopIteration
  File "C:\Users\Varun Jadia\Desktop\coding_assignments\ray\ray_venv\Lib\sit
e-packages\torch\utils\data\_utils\fetch.py", line 51, in fetch
    data = [self.dataset[idx] for idx in possibly_batched_index]
  File "C:\Users\Varun Jadia\AppData\Local\Temp\ipykernel_15472\1687281713.p
y", line 16, in __getitem__
    'input_ids': torch.tensor(item['input_ids'], dtype=torch.int64),
C:\Users\Varun Jadia\AppData\Local\Temp\ipykernel_15472\1687281713.py:16: Us
erWarning: Failed to initialize NumPy: _ARRAY_API not found (Triggered inter
nally at ..\torch\csrc\utils\tensor_numpy.cpp:84.)
  'input_ids': torch.tensor(item['input_ids'], dtype=torch.int64),
```

In [19]:

```
from transformers import AutoModelForTokenClassification
model = AutoModelForTokenClassification.from_pretrained('distilbert-base-unc
model.to('cpu')
model(input_ids=data['input_ids'], attention_mask=data['attention_mask'], la
```

```
Some weights of DistilBertForTokenClassification were not initialized from t
he model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
```

Out[19]:

```
TokenClassifierOutput(loss=tensor(1.0718, grad_fn=<NllLossBackward0>), logi
ts=tensor([[[-0.1770,  0.0028,  0.1541],
         [ 0.0462,  0.2013,  0.2575],
         [ 0.1894,  0.0635,  0.2393],
         ...,
         [-0.0151,  0.0837,  0.1055],
         [ 0.0402,  0.1049,  0.2192],
         [ 0.0572,  0.1422,  0.2047]]], grad_fn=<ViewBackward0>), hidden_st
ates=None, attentions=None)
```

An important part of the process is to define metrics to calculate the performance of our model. As ours is a classification task (we are classifying tokens into 1 of 3 entities), the metrics we will use are:

- Precision
- Recall
- F1 score
- Accuracy

We calculate these using a confusion matrix

In [20]:
```
# Define function to calculate precision, accuracy, f1score
def evaluate_token_classification(model, dataloader):
    model.to('cpu')
    model.eval()

    confusion = torch.zeros(3, 3, dtype=torch.long)

    with torch.no_grad():
        for batch in dataloader:
            input_ids = batch['input_ids'].to('cpu')
            attention_mask = batch['attention_mask'].to('cpu')
            labels = batch['labels']

            outputs = model(input_ids=input_ids, attention_mask=attention_ma
            logits = outputs.logits
            preds = torch.argmax(logits, dim=-1).cpu()

            for i in range(len(preds)):
                mask = attention_mask[i].cpu().bool()
                pred_tokens = preds[i][mask]
                label_tokens = labels[i][mask]

                for true_label, pred_label in zip(label_tokens, pred_tokens)
                    if not (true_label == -100 or pred_label == -100):
                        confusion[true_label, pred_label] += 1

    total_samples = confusion.sum().item()
    correct_predictions = confusion.diag().sum().item()
    accuracy = correct_predictions / total_samples if total_samples > 0 else

    class_metrics = {}
    for class_idx in range(3):
        true_positives = confusion[class_idx, class_idx].item()
        false_positives = confusion[:, class_idx].sum().item() - true_positi
        false_negatives = confusion[class_idx, :].sum().item() - true_positi

        precision = true_positives / (true_positives + false_positives) if (
        recall = true_positives / (true_positives + false_negatives) if (tru
        f1 = 2 * precision * recall / (precision + recall) if (precision + r

        class_metrics[class_idx] = {
            'precision': precision,
```

```
            'recall': recall,
            'f1': f1
        }

    # Prepare results
    results = {
        'class_metrics': class_metrics,
        'accuracy': accuracy
    }

    return results

evaluate_token_classification(model, dl)
```

Out[20]:

```
{'class_metrics': {0: {'precision': 0.0, 'recall': 0.0, 'f1': 0},
  1: {'precision': 0.26881720430107525,
   'recall': 0.7575757575757576,
   'f1': 0.3968253968253968},
  2: {'precision': 0.7222222222222222,
   'recall': 0.28888888888888886,
   'f1': 0.4126984126984127}},
 'accuracy': 0.3893129770992366}
```

In [21]:

```
# freeze bert parameters and only allow updates for classification head - th
for param in model.distilbert.parameters():
    param.requires_grad = False

for param in model.classifier.parameters():
    param.requires_grad = True
```

Now we create a training loop using the Ray Train framework, which allows for distributed training using a simple ScalingConfig, TrainerConfig - Ray dynamically splits the training set among workers, manages weight updates between workers, etc. to allow you to speed up training by running it over several cores. Recall that typically in pytorch, the training loop is run as a simple for loop. Here's an example:

```
for epoch in range(2): # number of times loop over train set
    running_loss = 0.0
    for i, data in enumerate(trainloader):
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward() # calculate gradient updates, i.e.,
backprop
        optimizer.step() # update weights

        # print statistics
```

```
            running_loss += loss.item()
            if i % 10 == 0:    # print every 10 mini-batches
                val_accuracy = eval_model(net, valloader)
                print(f'[{epoch + 1}, {i + 1:5d}] loss:
    {running_loss / 10:.3f}, val accuracy: {val_accuracy}')
                running_loss = 0.0

    print('Finished Training')
```

In [22]:

```
# Note that limit still returns a Dataset
train_set, val_set = tokenized_ds.train_test_split(test_size=0.30)
train_set.count(), val_set.count()
```

```
2025-03-01 19:30:11,728 INFO streaming_executor.py:108 -- Starting execution
of Dataset. Full logs are in C:\Users\VARUNJ~1\AppData\Local\Temp\ray\sessio
n_2025-03-01_19-28-07_605513_15472\logs\ray-data
2025-03-01 19:30:11,738 INFO streaming_executor.py:109 -- Execution plan of
Dataset: InputDataBuffer[Input] -> TaskPoolMapOperator[Map(parse_file)->MapB
atches(lemmatize_tokenize_and_align_labels)] -> AggregateNumRows[AggregateNu
mRows]
Running 0: 0.00 row [00:00, ? row/s]
- Map(parse_file)->MapBatches(lemmatize_tokenize_and_align_labels) 1: 0.00 r
ow [00:00, ? row/s]
- AggregateNumRows 2: 0.00 row [00:00, ? row/s]
2025-03-01 19:30:13,583 INFO streaming_executor.py:108 -- Starting execution
of Dataset. Full logs are in C:\Users\VARUNJ~1\AppData\Local\Temp\ray\sessio
n_2025-03-01_19-28-07_605513_15472\logs\ray-data
2025-03-01 19:30:13,589 INFO streaming_executor.py:109 -- Execution plan of
Dataset: InputDataBuffer[Input] -> TaskPoolMapOperator[Map(parse_file)->MapB
atches(lemmatize_tokenize_and_align_labels)]
Running 0: 0.00 row [00:00, ? row/s]
- Map(parse_file)->MapBatches(lemmatize_tokenize_and_align_labels) 1: 0.00 r
ow [00:00, ? row/s]
```

Out[22]: (28, 12)

In [23]:

```
import os
import tempfile
from transformers import AdamW, get_linear_schedule_with_warmup
from ray.train.torch import TorchTrainer
from ray.train import ScalingConfig, RunConfig, Checkpoint
import ray

def train_loop_per_worker(config):
    model_name = config["model_name"]
    num_labels = config["num_labels"]
    epochs = config["num_epochs"]
    batch_size = config["batch_size"]
    learning_rate = config["learning_rate"]

    train_examples, val_examples = [], []
    train_data = ray.train.get_dataset_shard("train")
    val_data = ray.train.get_dataset_shard("val")

    if train_data is None or val_data is None:
```

```
        raise ValueError("Dataset shard is None. Ensure dataset is passed cc

    train_examples, val_examples = [], []

    for batch in train_data.iter_batches():
        train_examples.extend(batch["tokenized_inputs"])

    for batch in val_data.iter_batches():
        val_examples.extend(batch["tokenized_inputs"])

    model = AutoModelForTokenClassification.from_pretrained(model_name, num_
    train_dataset = TokenizedDataset(train_examples)
    train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuf

    # Setup optimizer and scheduler
    optimizer = AdamW([p for p in model.parameters() if p.requires_grad], lr
    total_steps = len(train_dataloader) * epochs
    scheduler = get_linear_schedule_with_warmup(
        optimizer,
        num_warmup_steps=int(0.1 * total_steps),
        num_training_steps=total_steps
    )

    # Training loop
    for epoch in range(epochs):
        model.train()
        epoch_loss = 0

        for batch in train_dataloader:
            input_ids = batch['input_ids']
            attention_mask = batch['attention_mask']
            labels = batch['labels']

            # Forward pass
            outputs = model(input_ids=input_ids, attention_mask=attention_ma
            loss = outputs.loss # Note: CrossEntropy loss is automatically c

            # Backward pass, ensure you zero out gradients first
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            scheduler.step()
            epoch_loss += loss.item()

        train_accuracy = evaluate_token_classification(model, train_dataloac
        avg_loss = epoch_loss / len(train_dataloader)

        # Save checkpoint only from the rank 0 worker to prevent redundant c
        if ray.train.get_context().get_world_rank() == 0 and (epoch % config
            checkpoint_dir = os.path.join(config["checkpoint_dir"], f"epoch_
            os.makedirs(checkpoint_dir, exist_ok=True)
            torch.save(model.state_dict(), os.path.join(checkpoint_dir, "moc
            checkpoint = Checkpoint.from_directory(checkpoint_dir) # wrapper

            val_dataset = TokenizedDataset(val_examples)
            val_dataloader = DataLoader(val_dataset, batch_size=batch_size,
```

```
            val_accuracy = evaluate_token_classification(model, val_dataloac

            ray.train.report({"loss": avg_loss, "val_accuracy": val_accuracy
        else:
            ray.train.report({"loss": avg_loss})
```

In [ ]:
```
# Define training configuration
train_config = {
    "model_name": "distilbert-base-uncased",
    "num_labels": 3,  # B, I, O
    "num_epochs": 2,
    "batch_size": 10, # pytorch batch size, keep small due to system constra
    "learning_rate": 3e-5,
    "checkpoint_dir": 'C:\\Users\\Varun Jadia\\Desktop\\coding_assignments\\
    "checkpoint_freq": 1 # 1 = save checkpoint every epoch
}

scaling_config = ScalingConfig(
    num_workers=1, # scale up as necessary
    use_gpu=False,
)

trainer = TorchTrainer(
    train_loop_per_worker=train_loop_per_worker,
    train_loop_config=train_config,
    scaling_config=scaling_config,
    datasets={"train": train_set, "val": val_set},
    run_config=RunConfig(
        name="biobert_ner_training",
        storage_path='C:\\Users\\Varun Jadia\\Desktop\\coding_assignments\\r
    ),
)

results = trainer.fit()
print(f"Training complete. Results: {results}")
```

Ray also has a checkpointing feature that allows us to save the model’s state at regular intervals. In the training loop, we save checkpoints only on the main worker (world_rank == 0) at a specified frequency (checkpoint_freq). The model’s state dictionary is stored in the configured directory, and Ray’s Checkpoint.from_directory() registers it with Ray Train for tracking. This ensures efficient checkpointing without redundant saves across distributed workers.

In [26]:
```
final_checkpoint = results.checkpoint
best_model = AutoModelForTokenClassification.from_pretrained(
    "distilbert-base-uncased",
    num_labels=3
)
best_model.load_state_dict(torch.load(os.path.join(final_checkpoint.path, "m
```

```
Some weights of DistilBertForTokenClassification were not initialized from t
he model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
```

Out[26]:

```
<All keys matched successfully>
```

Get test data metrics

We now evaluate our finetuned model on test data. To do this, we build a test dataset using our previously defined functions, now applied to test.tsv files

In [27]:

```
test_file_paths = glob.glob("./NERData/**/test.tsv", recursive=True)
test_ds = ray.data.from_items([{"file_path": path} for path in test_file_pat
processed_test_ds = test_ds.map(parse_file)
tokenized_test_ds = processed_test_ds.map_batches(lemmatize_tokenize_and_ali
tokenized_test_ds.count()
```

```
2025-03-01 19:48:51,392 INFO streaming_executor.py:108 -- Starting execution
of Dataset. Full logs are in C:\Users\VARUNJ~1\AppData\Local\Temp\ray\sessio
n_2025-03-01_19-28-07_605513_15472\logs\ray-data
2025-03-01 19:48:51,395 INFO streaming_executor.py:109 -- Execution plan of
Dataset: InputDataBuffer[Input] -> TaskPoolMapOperator[Map(parse_file)->MapB
atches(lemmatize_tokenize_and_align_labels)] -> AggregateNumRows[AggregateNu
mRows]
Running 0: 0.00 row [00:00, ? row/s]
- Map(parse_file)->MapBatches(lemmatize_tokenize_and_align_labels) 1: 0.00 r
ow [00:00, ? row/s]
- AggregateNumRows 2: 0.00 row [00:00, ? row/s]
```

Out[27]:

```
40
```

In [28]:

```
best_model.eval()
test_examples = []
for batch in tokenized_test_ds.iter_batches():
    test_examples.extend(batch["tokenized_inputs"])

test_dataset = TokenizedDataset(test_examples)
test_dataloader = DataLoader(test_dataset, batch_size=2, shuffle=True)
evaluate_token_classification(best_model, test_dataloader)
```

```
2025-03-01 19:49:25,613 INFO streaming_executor.py:108 -- Starting execution
of Dataset. Full logs are in C:\Users\VARUNJ~1\AppData\Local\Temp\ray\sessio
n_2025-03-01_19-28-07_605513_15472\logs\ray-data
2025-03-01 19:49:25,614 INFO streaming_executor.py:109 -- Execution plan of
Dataset: InputDataBuffer[Input] -> TaskPoolMapOperator[Map(parse_file)->MapB
atches(lemmatize_tokenize_and_align_labels)]
Running 0: 0.00 row [00:00, ? row/s]
- Map(parse_file)->MapBatches(lemmatize_tokenize_and_align_labels) 1: 0.00 r
ow [00:00, ? row/s]
```

Out[28]:
```
{'class_metrics': {0: {'precision': 0, 'recall': 0.0, 'f1': 0},
  1: {'precision': 0, 'recall': 0.0, 'f1': 0},
  2: {'precision': 0.7901754385964912,
   'recall': 1.0,
   'f1': 0.8827910623284986}},
 'accuracy': 0.7901754385964912}
```

Note that model accuracy/other metrics are likely to be bad if trained on a limited number of samples/few epochs

Using Ray tune to train learning rate

Hyperparameter tuning is a standard part of any ML workflow - Ray also provides an interface to do this in a similar fashion to our training setup above. For this example, we'll find the optimal learning rate for our model above by giving Ray a search space to look over. Some key features of Ray tune being used here:

- ASHAScheduler: controls how trials are terminated early to save computational resources
- BayesOptSearch: This is the search algorithm we use here by Ray to select the next lr value to try. Generally better than a simple grid search

In [ ]:
```python
from ray import tune
from ray.tune.schedulers import ASHAScheduler
from ray.tune.search.bayesopt import BayesOptSearch

def trial_dirname_creator(trial):
    return f"trial_{trial.trial_id}"

def tune_learning_rate(num_samples):
    search_space = {
        "learning_rate": tune.loguniform(1e-5, 1e-3),
    }

    base_config = {k: v for k, v in train_config.items() if k != "learning_r
    tune_config = {**base_config, **search_space}

    scheduler = ASHAScheduler(
        max_t=train_config["num_epochs"],
        grace_period=1,
        reduction_factor=2,
        metric="val_accuracy",
        mode="max"
    )

    search_alg = BayesOptSearch(metric="loss", mode="min")

    def trainable(config):
        # The config passed to this function will include the sampled learni
        trainer = TorchTrainer(
            train_loop_per_worker=train_loop_per_worker,
            train_loop_config=config,
```

```python
            scaling_config=scaling_config,
            datasets={"train": train_set, "val": val_set},
        )
        result = trainer.fit()
        return result

    tuner = tune.Tuner(
        trainable=trainable,
        param_space=tune_config,
        tune_config=tune.TuneConfig(
            num_samples=num_samples,
            scheduler=scheduler,
            search_alg=search_alg,
            trial_dirname_creator=trial_dirname_creator
        ),
        run_config=RunConfig(
            name="learning_rate_tuning",
            storage_path="C:\\Users\\Varun Jadia\\Desktop\\coding_assignment
        )
    )

    results = tuner.fit()
    best_result = results.get_best_result(metric="val_accuracy", mode="max")
    best_config = best_result.config
    best_checkpoint = best_result.checkpoint

    return best_config, best_checkpoint

# we try only 2 different learning rates in this example...
best_config, best_checkpoint = tune_learning_rate(num_samples=2)
```

In [ ]:

```python

```