1. Load Datasets

Loading test, train and validation sets into pandas dataframe

```
import pandas as pd

df_train = pd.read_json('data/train.json', lines=True)
df_test = pd.read_json('data/test.json', lines=True)
df_validation = pd.read_json('data/validation.json', lines=True)
```

2. Exploratory Data Analysis (Sayeed, Jui)

Analyzing training dataset

```
df_train.info()
```

```
<class 'pandas.DataFrame'>
RangeIndex: 1112 entries, 0 to 1111
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   quality_checked  1112 non-null   object
 1   text             1112 non-null   str
 2   task             1112 non-null   str
 3   meta             1112 non-null   object
 4   doc_id           1112 non-null   str
 5   dataset_type     1112 non-null   str
 6   annotator_id     1112 non-null   str
 7   entity_mentions  1112 non-null   object
dtypes: object(3), str(5)
memory usage: 69.6+ KB
```

Observed no null values

```
df_train.head()
```

	quality_checked	text	task	meta	doc_id	dataset_type	annotator_id	entity_mentions
0	[]	PROCEDURE\n\nThe case originated in an applica...	Task: Annotate the document to anonymise the f...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194	train	annotator1	[{'confidential_status': 'NOT_CONFIDENTIAL', '...
1	[]	PROCEDURE\n\nThe case originated in an applica...	Task: Annotate the document to anonymise the f...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194	train	annotator2	[{'confidential_status': 'NOT_CONFIDENTIAL', '...
		PROCEDURE\n\nThe case	Task: Annotate the	{'applicant':	001			[{'confidential_status':

Observation: We do not need dataset_type because the test, train and validation files are already separate. We also do not need the columns quality_checked, annotator_id.

Dropping unrequired columns.

```
df_train.drop(columns=['quality_checked', 'dataset_type', 'annotator_id'])
```

	text	task	meta	doc_id	entity_mentions
0	PROCEDURE\n\nThe case originated in an applica...	Task: Annotate the document to anonymise the f...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194	[{'confidential_status': 'NOT_CONFIDENTIAL', '...
1	PROCEDURE\n\nThe case originated in an applica...	Task: Annotate the document to anonymise the f...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194	[{'confidential_status': 'NOT_CONFIDENTIAL', '...
2	PROCEDURE\n\nThe case originated in an applica...	Task: Annotate the document to anonymise the f...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194	[{'confidential_status': 'NOT_CONFIDENTIAL', '...
3	PROCEDURE\n\nThe case originated in an applica...	Task: Annotate the document to anonymise the f...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194	[{'confidential_status': 'NOT_CONFIDENTIAL', '...
4	PROCEDURE\n\nThe case originated in an applica...	Task: Annotate the document to anonymise the f...	{'applicant': 'D. Stępniak', 'articles': [91, ...	001-84741	[{'confidential_status': 'NOT_CONFIDENTIAL', '...
...	...	...	...	...	...
1107	PROCEDURE\n\nThe case originated in an applica...	Task: Annotate the document to anonymise the f...	{'applicant': 'Helmut Ludescher', 'articles': ...	001-60002	[{'confidential_status': 'NOT_CONFIDENTIAL', '...
1108	PROCEDURE\n\nThe case originated in an applica...	Task: Annotate the document to anonymise the f...	{'applicant': 'J. Peter', 'articles': [91, 34,...	001-146353	[{'confidential_status': 'NOT_CONFIDENTIAL', '...

Analyzing task column

```
df_train['task'].unique()
```

```
<StringArray>
[       'Task: Annotate the document to anonymise the following person: Henrik Hasslund',
            'Task: Annotate the document to anonymise the following person: D. Stępniak',
         'Task: Annotate the document to anonymise the following person: Nusret Amutgan',
           'Task: Annotate the document to anonymise the following person: Mustafa Sarı',
      'Task: Annotate the document to anonymise the following person: Dariusz Karwowski',
         'Task: Annotate the document to anonymise the following person: İlhan Karakurt',
        'Task: Annotate the document to anonymise the following person: Artur Warsiński',
   'Task: Annotate the document to anonymise the following person: Włodzimierz Majewski',
         'Task: Annotate the document to anonymise the following person: İlhami Erseven',
            'Task: Annotate the document to anonymise the following person: Semir Güzel',
 ...
       'Task: Annotate the document to anonymise the following person: Zekeriya Karaman',
 'Task: Annotate the document to anonymise the following person: Leandro Sanchez-Reisse',
          'Task: Annotate the document to anonymise the following person: Rémi Bertuzzi',
    'Task: Annotate the document to anonymise the following person: Andrés López Elorza',
       'Task: Annotate the document to anonymise the following person: Frédéric Foucher',
           'Task: Annotate the document to anonymise the following person: Faruk Ereren',
       'Task: Annotate the document to anonymise the following person: Helmut Ludescher',
               'Task: Annotate the document to anonymise the following person: J. Peter',
  'Task: Annotate the document to anonymise the following person: Christopher Ian Scott',
       'Task: Annotate the document to anonymise the following person: Yiannis Kyriakou']
Length: 1008, dtype: str
```

Observation: we don't need the task column

Finding out how many unique values are there in text column and doc_id column. Making sure they match.

```
len(df_train['text'].unique())
```

```
1014
```

```
len(df_train['doc_id'].unique())
```

```
1014
```

Observation: We have 1014 unique values for documents but the dataset has 1112 entries. So there might be duplicates.

```
df_train['meta'][0]
```

```
{'applicant': 'Henrik Hasslund',
 'articles': [91, 34, 54, 34, 93],
 'countries': 'DNK',
 'legal_branch': 'CHAMBER',
 'year': 2008}
```

Observation: We may be able to reserve this column for later evaluation. Might be helpful to find out if our model struggles with region specific names, or has a bias, etc.

```
df_train_meta = df_train[['text', 'meta', 'doc_id']].copy()
df_train_meta.head()
```

	text	meta	doc_id
0	PROCEDURE\n\nThe case originated in an applica...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194
1	PROCEDURE\n\nThe case originated in an applica...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194
2	PROCEDURE\n\nThe case originated in an applica...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194
3	PROCEDURE\n\nThe case originated in an applica...	{'applicant': 'Henrik Hasslund', 'articles': [...	001-90194
4	PROCEDURE\n\nThe case originated in an applica...	{'applicant': 'D. Stępniak', 'articles': [91, ...	001-84741

```
df_train_meta.to_csv('data/metadata/train_meta.csv')
```

```
df_test_meta = df_test[['text', 'meta', 'doc_id']].copy()
df_test_meta.to_csv('data/metadata/test_meta.csv')

df_validation_meta = df_validation[['text', 'meta', 'doc_id']].copy()
df_validation_meta.to_csv('data/metadata/validation_meta.csv')
```

Exploring entity mentions column

```
df_train['entity_mentions'][1]
```


```
[{'confidential_status': 'NOT_CONFIDENTIAL',
  'edit_type': 'check',
  'end_offset': 62,
  'entity_id': '001-90194_a2_e1',
  'entity_mention_id': '001-90194_a2_em1',
  'entity_type': 'CODE',
  'identifier_type': 'QUASI',
  'related_mentions': None,
  'span_text': '36244/06',
  'start_offset': 54},
 {'confidential_status': 'NOT_CONFIDENTIAL',
  'edit_type': 'correct',
  'end_offset': 94,
  'entity_id': '001-90194_a2_e2',
  'entity_mention_id': '001-90194_a2_em2',
  'entity_type': 'ORG',
  'identifier_type': 'QUASI',
  'related_mentions': None,
  'span_text': 'Kingdom of Denmark',
  'start_offset': 76},
 {'confidential_status': 'NOT_CONFIDENTIAL',
  'edit_type': 'check',
  'end_offset': 242,
  'entity_id': '001-90194_a2_e3',
  'entity_mention_id': '001-90194_a2_em3',
  'entity_type': 'DEM',
  'identifier_type': 'QUASI',
  'related_mentions': None,
  'span_text': 'Danish',
  'start_offset': 236},
 {'confidential_status': 'NOT_CONFIDENTIAL',
  'edit_type': 'check',
  'end_offset': 271,
  'entity_id': '001-90194_a2_e4',
  'entity_mention_id': '001-90194_a2_em4',
  'entity_type': 'PERSON',
  'identifier_type': 'DIRECT',
  'related_mentions': None,
  'span_text': 'Mr Henrik Hasslund',
  'start_offset': 253},
 {'confidential_status': 'NOT_CONFIDENTIAL',
  'edit_type': 'check',
  'end_offset': 308,
  'entity_id': '001-90194_a2_e5',
  'entity_mention_id': '001-90194_a2_em5',
  'entity_type': 'DATETIME',
  'identifier_type': 'QUASI',
  'related_mentions': None,
  'span_text': '31 August 2006',
  'start_offset': 294},
 {'confidential_status': 'NOT_CONFIDENTIAL',
  'edit_type': 'check',
  'end_offset': 357,
  'entity_id': '001-90194_a2_e6',
  'entity_mention_id': '001-90194_a2_em6',
  'entity_type': 'PERSON',
  'identifier_type': 'QUASI',
  'related_mentions': None,
```


```
import json
df_train_exploded = df_train.explode('entity_mentions')
entities_flat = pd.json_normalize(df_train_exploded['entity_mentions'])
```

```
df_train_entities = pd.concat([df_train_exploded[['doc_id']].reset_index(drop=True), entities_flat.reset_index(drop=True)], axis=1)

df_train_entities.head()
```

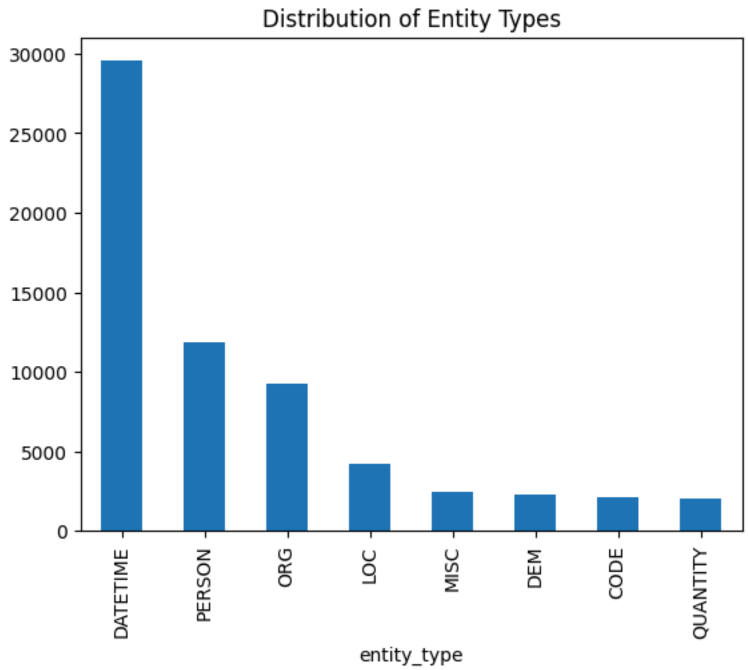
	doc_id	confidential_status	edit_type	end_offset	entity_id	entity_mention_id	entity_type	identifier_type	related_mentions	span_tex
0	001-90194	NOT_CONFIDENTIAL	check	62	001-90194_a1_e1	001-90194_a1_em1	CODE	DIRECT	None	36244/0
1	001-90194	NOT_CONFIDENTIAL	correct	94	001-90194_a1_e2	001-90194_a1_em2	ORG	NO_MASK	None	Kingdon c Denmar
2	001-90194	NOT_CONFIDENTIAL	check	242	001-90194_a1_e3	001-90194_a1_em3	DEM	NO_MASK	None	Danisl

Observation: We have start_offset, end_offset and entity_type. We need to extract this data to create a token and tags for finetuning DistilliBERT model.

Checking the counts of entities for filtered set where identifier_type is not NO_MASK

```
print("Entity Type Counts:")
entity_type_stats = df_train_entities[df_train_entities['identifier_type'] != 'NO_MASK']['entity_type'].value_counts()
entity_type_stats.plot(kind='bar', title='Distribution of Entity Types')
```

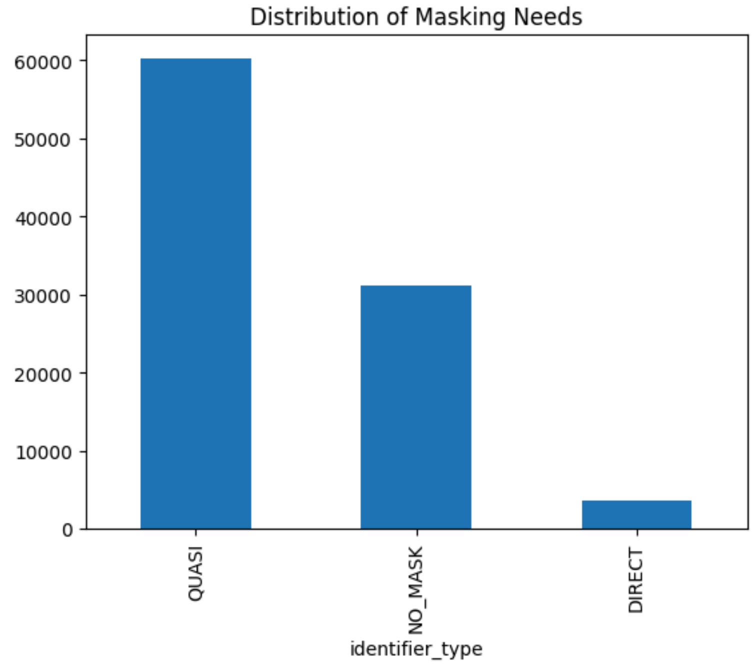
```
Entity Type Counts:
<Axes: title={'center': 'Distribution of Entity Types'}, xlabel='entity_type'>
```



Certain imbalance of data is seen here where datetime entities are much higher in count than quantity or code

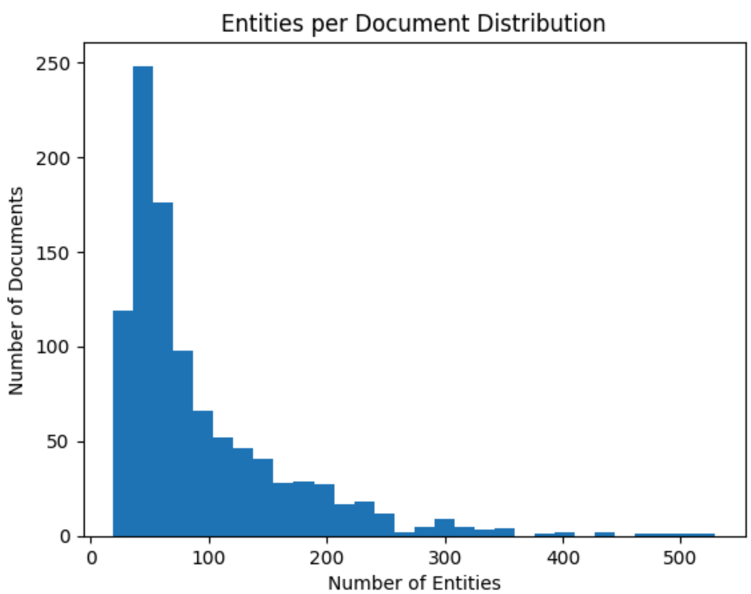
```
# Check masking requirements
mask_stats = df_train_entities['identifier_type'].value_counts()
mask_stats.plot(kind='bar', title='Distribution of Masking Needs')
```

```
<Axes: title={'center': 'Distribution of Masking Needs'}, xlabel='identifier_type'>
```




```
entities_per_doc = df_train_entities.groupby('doc_id').size().sort_values(ascending=False)
entities_per_doc.plot(kind='hist', bins=30, title='Entities per Document Distribution', xlabel='Number of Entities', ylabel='Number of Documer
```

```
<Axes: title={'center': 'Entities per Document Distribution'}, xlabel='Number of Entities', ylabel='Number of Documents'>
```




```
print(f"Average entities per document: {entities_per_doc.mean():.2f}")
```

```
Average entities per document: 93.72
```

```python
import pandas as pd

def read_data(traindata, testdata, validationdata):
    df_train = pd.read_json(traindata, lines=True)
    df_test = pd.read_json(testdata, lines=True)
    df_validation = pd.read_json(validationdata, lines=True)
    return df_train, df_test, df_validation
```

3. Data pre-processing (Jui)

Converting offests to list

```python
def convert_offsets_to_lists(row):
    text = row['text']
    entities = row['entity_mentions']

    # create character-level map
```

```
    char_tags = ["O"] * len(text)

    for ent in entities:
        # Filter 'NO_MASK' entities
        if ent.get('identifier_type') == 'NO_MASK':
            continue

        start, end = ent['start_offset'], ent['end_offset']
        label = ent['entity_type']

        # fill character-level map
        if start < len(text) and end <= len(text):
            char_tags[start] = f"B-{label}" # beginning of entity
            for i in range(start+1, end):
                char_tags[i] = f"I-{label}" # inside entity

    # convert character map to word - tag
    tokens = text.split()
    ner_tags = []

    cursor = 0
    for token in tokens:
        # advance cursor to the start of word (skipping spaces)
        while cursor < len(text) and text[cursor].isspace():
            cursor += 1

        # tag of the word is the tag of its first character
        if cursor < len(text):
            ner_tags.append(char_tags[cursor])
            cursor += len(token)
        else:
            ner_tags.append("O")

    return {"tokens": tokens, "ner_tags": ner_tags}
```

```
Start coding or generate with AI.
```

```
from datasets import Dataset

df_train, df_test, df_validation = read_data('data/train.json', 'data/test.json', 'data/validation.json')

# converting Pandas to Hugging Face Dataset
hf_train = Dataset.from_pandas(df_train)
train_processed = hf_train.map(convert_offsets_to_lists)

# Quick Check:
print(train_processed[0]['tokens'])
print(train_processed[0]['ner_tags'])
```

```
/home/fred/anaconda3/envs/LDR-NER/lib/python3.14/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipy
  from .autonotebook import tqdm as notebook_tqdm
Map: 100%|██████████| 1112/1112 [00:03<00:00, 302.96 examples/s]['PROCEDURE', 'The', 'case', 'originated', 'in', 'an', 'application', '(no.',
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-CODE', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'C
```

Create label mappings from train set to be used to pass when training the model, same mappings would be used during tokenization of test and validation sets as well

```
# extracting unique tags from training data
unique_tags = set(tag for row in train_processed for tag in row['ner_tags'])
label_list = sorted(list(unique_tags)) # e.g., ['B-LOC', 'B-PER', 'I-PER', 'O']

# createing maps
label2id = {label: i for i, label in enumerate(label_list)}
id2label = {i: label for i, label in enumerate(label_list)}

print(f"Number of labels: {len(label_list)}")
print(label2id)
```

```
Number of labels: 17
{'B-CODE': 0, 'B-DATETIME': 1, 'B-DEM': 2, 'B-LOC': 3, 'B-MISC': 4, 'B-ORG': 5, 'B-PERSON': 6, 'B-QUANTITY': 7, 'I-CODE': 8, 'I-DATETIME': 9,
```

Using AutoTokenizer to handle sub-words and align new tags. Defining tokenization function to be used various versions of BERT

```
def tokenize_and_align(examples, tokenizer):
  '''Takes the tokenizer object as input and operated on a row in huggingface dataset object'''
  # split words into sub-words
  tokenized_inputs = tokenizer(examples["tokens"], truncation=True, is_split_into_words=True)
```

```
    labels = []
    for i, label in enumerate(examples["ner_tags"]):
        word_ids = tokenized_inputs.word_ids(batch_index=i) #supported only for fast tokenizers
        previous_word_idx = None
        label_ids = []
        for word_idx in word_ids:
            if word_idx is None:
                # special tokens like [CLS] get -100 (ignored)
                label_ids.append(-100)
            elif word_idx != previous_word_idx:
                # first piece of a word gets the real label ID
                label_ids.append(label2id[label[word_idx]]) #using map created from training set
            else:
                # subsequent pieces (e.g., "##lor") get -100 (ignored)
                label_ids.append(-100)
            previous_word_idx = word_idx
        labels.append(label_ids)

    tokenized_inputs["labels"] = labels
    return tokenized_inputs
```

Writing a script for processing test and validation sets

```
def preprocess_data(df, tokenizer):
  '''Takes a pandas dataframe and a tokenizer object and returns a tokenized huggingface dataset object ready to be passed into BERT for trair
  '''
  # converting Pandas to Hugging Face Dataset
  hf = Dataset.from_pandas(df)
  processed = hf.map(convert_offsets_to_lists)
  #tokenize
  tokenized = processed.map(tokenize_and_align, batched=True, fn_kwargs={"tokenizer": tokenizer})

  return tokenized
```

Importing AutoTokenizer to use various BERT tokenizers

```
from transformers import AutoTokenizer
```

4. Bi-LSTM (Jui)

Using DistilBERT tokenizer to keep the comparison fair

```
tokenizer_distilbert = AutoTokenizer.from_pretrained("distilbert-base-uncased")
```

```
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
```

```
train_distilbert = preprocess_data(df_train ,tokenizer_distilbert)
```

```
Map: 100%|██████████| 1112/1112 [00:03<00:00, 342.03 examples/s]
Map: 100%|██████████| 1112/1112 [00:08<00:00, 131.21 examples/s]
```

```
test_distilbert = preprocess_data(df_test, tokenizer_distilbert)
validation_distilbert = preprocess_data(df_validation, tokenizer_distilbert)
```

```
Map: 100%|██████████| 555/555 [00:00<00:00, 568.19 examples/s]
Map: 100%|██████████| 555/555 [00:02<00:00, 217.83 examples/s]
Map: 100%|██████████| 541/541 [00:00<00:00, 587.80 examples/s]
Map: 100%|██████████| 541/541 [00:02<00:00, 230.29 examples/s]
```

Importing pyTorch

```
import torch
import torch.nn as nn
```

```
#setup GPU availability
if torch.backends.mps.is_available():
    device = torch.device('mps') #mac m3 chip integrated gpu
    print(f"MPS device found: {device}")
elif torch.cuda.is_available():
    device = torch.device('cuda')
    print(f"GPU device found: {device}")
else:
    device = torch.device('cpu')
    print(f"No GPU device found. Falling back to {device}")
```

```
MPS device found: mps
```

Defining the Bi-LSTM model for NER

```
class bilstm_nre(nn.Module):
  def __init__(self, vocab_size, num_labels, embed_dim=128, hidden_dim=256, weight_tensor=None, dropout_rate=None):
      super().__init__()

      #setting up a custom embedding layer
      #seting padding index to 0 the embedding model from learning vectors for padding
      self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)

      #bi-lstm layer
      self.lstm = nn.LSTM(embed_dim, hidden_dim, batch_first=True, bidirectional=True)

      #dropout
      if dropout_rate is not None:
        self.dropout = nn.Dropout(dropout_rate)

      #output layer (2*hidden_dim for bi-lstm)
      self.classifier = nn.Linear(hidden_dim * 2, num_labels)

      #set loss function (ignore_index=-100 because we set masking to -100 in previous function)
      if weight_tensor is not None:
        self.loss_fct = nn.CrossEntropyLoss(ignore_index=-100, weight=weight_tensor)
      else:
        self.loss_fct = nn.CrossEntropyLoss(ignore_index=-100)

  #forward pass
  def forward(self, input_ids, labels=None):
      #embed
      embeds = self.embedding(input_ids)

      #lstm forward
      lstm_out, _ = self.lstm(embeds)

      logits = self.classifier(lstm_out)

      #loss calculation
      loss = None
      if labels is not None:
          # Flatten the tensors so we can check every token at once
          # logits shape: (batch * seq_len, num_labels)
          # labels shape: (batch * seq_len)
          loss = self.loss_fct(logits.view(-1, logits.shape[-1]), labels.view(-1))

      return loss, logits
```

batch pre-processing function to prepare dataloaders

```
def collate_fn(batch):
    # convert batch to tensors
    input_ids = [torch.tensor(item['input_ids']) for item in batch]
    labels = [torch.tensor(item['labels']) for item in batch]

    # padding inputs with 0(blank space), labels with -100
    input_ids = torch.nn.utils.rnn.pad_sequence(input_ids, batch_first=True, padding_value=0)
    labels = torch.nn.utils.rnn.pad_sequence(labels, batch_first=True, padding_value=-100)

    return input_ids.to(device), labels.to(device)
```

```
from torch.utils.data import DataLoader

train_loader = DataLoader(train_distilbert, batch_size=16, shuffle=True, collate_fn=collate_fn)
```

```
test_loader = DataLoader(test_distilbert, batch_size=16, shuffle=True, collate_fn=collate_fn)
validation_loader = DataLoader(validation_distilbert, batch_size=16, shuffle=True, collate_fn=collate_fn)
```

Tracking model performance at every epoch

```
#using huggingface evaluate library for NER evaluation
# !pip install evaluate seqeval
```

```
Collecting evaluate
  Downloading evaluate-0.4.6-py3-none-any.whl.metadata (9.5 kB)
Collecting seqeval
  Downloading seqeval-1.2.2.tar.gz (43 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/43.6 kB ? eta -:--:--
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.6/43.6 kB 2.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: datasets>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from evaluate) (4.0.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from evaluate) (2.0.2)
Requirement already satisfied: dill in /usr/local/lib/python3.12/dist-packages (from evaluate) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (from evaluate) (2.2.2)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.12/dist-packages (from evaluate) (2.32.4)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.12/dist-packages (from evaluate) (4.67.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from evaluate) (3.6.0)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.12/dist-packages (from evaluate) (0.70.16)
Requirement already satisfied: fsspec>=2021.05.0 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]>=2021.05.0->evaluate) (2025.3.0)
Requirement already satisfied: huggingface-hub>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from evaluate) (0.36.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from evaluate) (25.0)
Requirement already satisfied: scikit-learn>=0.21.3 in /usr/local/lib/python3.12/dist-packages (from seqeval) (1.6.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from datasets>=2.0.0->evaluate) (3.20.3)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.12/dist-packages (from datasets>=2.0.0->evaluate) (18.1.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from datasets>=2.0.0->evaluate) (6.0.3)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.12/dist-packages (from fsspec[http]>=2021.05.0->evaluate) (
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.7.0->evaluate) (4
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.7.0->evaluate) (1.2.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->evaluate) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->evaluate) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->evaluate) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->evaluate) (2026.1.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.21.3->seqeval) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.21.3->seqeval) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.21.3->seqeval) (3.6.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas->evaluate) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas->evaluate) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas->evaluate) (2025.3)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021.05
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=202
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2021
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=202
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas->evaluate) (1.17.0)
Downloading evaluate-0.4.6-py3-none-any.whl (84 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 6.9 MB/s eta 0:00:00
Building wheels for collected packages: seqeval
  Building wheel for seqeval (setup.py) ... done
  Created wheel for seqeval: filename=seqeval-1.2.2-py3-none-any.whl size=16162 sha256=b7cf57ae908c0f9adff6cf5bb8ddb236c47ede6d12addf74a96d8bde
  Stored in directory: /root/.cache/pip/wheels/5f/b8/73/0b2c1a76b701a677653dd79ece07cfabd7457989dbfbdcd8d7
Successfully built seqeval
Installing collected packages: seqeval, evaluate
Successfully installed evaluate-0.4.6 seqeval-1.2.2
```

```
import matplotlib.pyplot as plt
import evaluate
import numpy as np
```

Function to evaluate after every epoch

```
seqeval = evaluate.load("seqeval")

def evaluate_epoch(model, dataloader, label_list):
    model.eval() # Set to evaluation mode

    all_preds = []
    all_labels = []
    total_val_loss = 0

    with torch.no_grad():
        for batch_ids, batch_labels in dataloader:
            batch_ids = batch_ids.to(device)
            batch_labels = batch_labels.to(device)

            # Forward pass
            loss, logits = model(batch_ids, batch_labels)

            # test loss
            total_val_loss += loss.item()
```

```
            # Get predictions (argmax)
            preds = torch.argmax(logits, dim=-1).cpu().numpy()
            labels = batch_labels.cpu().numpy()

            all_preds.extend(preds)
            all_labels.extend(labels)

    #calculate validation loss
    avg_val_loss = total_val_loss / len(dataloader)

    # convert IDs back to Tags (removing -100)
    decoded_preds = [
        [label_list[p] for (p, l) in zip(pred, label) if l != -100]
        for pred, label in zip(all_preds, all_labels)
    ]
    decoded_labels = [
        [label_list[l] for (p, l) in zip(pred, label) if l != -100]
        for pred, label in zip(all_preds, all_labels)
    ]

    # compute metrics using seqeval (Strict Entity-Level scoring)
    results = seqeval.compute(predictions=decoded_preds, references=decoded_labels)

    return {
        "val_loss": avg_val_loss,
        "accuracy": results["overall_accuracy"],
        "precision": results["overall_precision"],
        "recall": results["overall_recall"],
        "f1": results["overall_f1"]
    }
```

Function for training loop with evaluation metric after each epoch

```
def train_eval_lstm(model, optimizer, n_epoches = 5, early_stopping=False):
  print("Starting Bi-LSTM for NER training...")

  history = {
      "train_loss": [],
      "val_loss": [],
      "accuracy": [],
      "precision": [],
      "recall": [],
      "f1": []
  }

  patience = 3
  patience_counter = 0
  best_val_loss = np.inf #initializing to infinite so that the first loss is always an improvement

  for epoch in range(n_epoches):
      # train
      model.train()
      total_loss = 0

      for batch_ids, batch_labels in train_loader:
          optimizer.zero_grad()
          loss, logits = model(batch_ids, batch_labels)
          loss.backward()
          optimizer.step()
          total_loss += loss.item()

      avg_train_loss = total_loss / len(train_loader)

      # validation on test set to calculate metrics
      metrics = evaluate_epoch(model, validation_loader, label_list)

      # tracking history of metrics
      history["train_loss"].append(avg_train_loss)
      history["val_loss"].append(metrics["val_loss"])
      history["accuracy"].append(metrics["accuracy"])
      history["precision"].append(metrics["precision"])
      history["recall"].append(metrics["recall"])
      history["f1"].append(metrics["f1"])

      print(f"Epoch {epoch+1}/{n_epoches} | "
            f"Train Loss: {avg_train_loss:.4f} | "
            f"Val Loss: {metrics['val_loss']:.4f} | "
            f"Val Recall: {metrics['recall']:.4f} | "
            f"Val Precision: {metrics['precision']:.4f} |"
            f"Val F1: {metrics['f1']:.4f} |"
```

```
            f"Val Accuracy: {metrics['accuracy']:.4f}")

        #early stopping
        if early_stopping == True:
          epoch_val_loss = metrics["val_loss"]
          if epoch_val_loss < best_val_loss:
              # if validation loss is improving/decreasing
              print(f"Validation Loss improved from {best_val_loss:.4f} to {epoch_val_loss:.4f}.")
              best_val_loss = epoch_val_loss
              patience_counter = 0  # reset the counter

          else:
              # validation loss increasing
              patience_counter += 1
              print(f"No improvement in validation loss. Patience: {patience_counter}")

              if patience_counter >= patience:
                  print("Early Stopping triggered! Training stopped.")
                  break # early stop


    print("Training complete!")
    return history
```

function for plotting the evaluation metrics vs epoch

```
def plot_training_metrics(history):
    epochs_range = range(1, len(history['train_loss']) + 1)

    plt.figure(figsize=(18, 6))

    # training vs validation loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, history['train_loss'], 'r-o', label='Training Loss')
    plt.plot(epochs_range, history['val_loss'], 'b-o', label='Validation Loss')
    plt.title('Training Loss vs Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    # R vs P vs A vs F1
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, history['recall'], 'b-o', label='Val Recall')
    plt.plot(epochs_range, history['f1'], 'g-o', label='Val F1 Score')
    plt.plot(epochs_range, history['accuracy'], 'k-x', label='Accuracy', linestyle='--')
    plt.plot(epochs_range, history['precision'], 'r-o', label='Precision')
    plt.title('Validation R vs P vs A vs F1')
    plt.xlabel('Epochs')
    plt.ylabel('Score')
    plt.ylim(0, 1.0) # y-axis to 0-100%
    plt.legend(loc='lower right')
    plt.grid(True)

    plt.tight_layout()
    plt.show()
```

Training & Evaluation LSTM model v1

```
# vocab_size is 30522 for DistilBERT
# num_labels is len(label_list)
lstm_nre_v1 = bilstm_nre(30522, len(label_list), embed_dim=128, hidden_dim=256).to(device)
optimizer_v1 = torch.optim.Adam(lstm_nre_v1.parameters(), lr=1e-4)
```

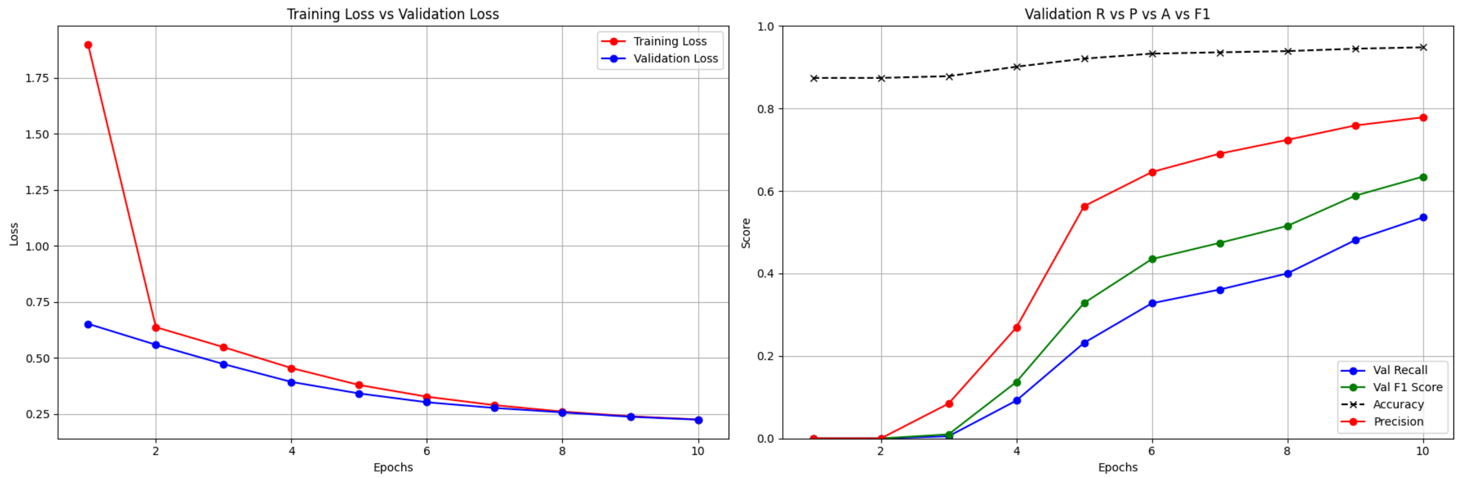
```
history_v1 = train_eval_lstm(lstm_nre_v1, optimizer_v1, n_epoches = 10)
```

```
Starting Bi-LSTM for NER training...
/opt/anaconda3/envs/hf-venv/lib/python3.11/site-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-define
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/hf-venv/lib/python3.11/site-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-define
  _warn_prf(average, modifier, msg_start, len(result))
Epoch 1/10 | Train Loss: 1.8969 | Val Loss: 0.6523 | Val Recall: 0.0000 | Val Precision: 0.0000 |Val F1: 0.0000 |Val Accuracy: 0.8740
Epoch 2/10 | Train Loss: 0.6376 | Val Loss: 0.5592 | Val Recall: 0.0000 | Val Precision: 0.0000 |Val F1: 0.0000 |Val Accuracy: 0.8740
Epoch 3/10 | Train Loss: 0.5484 | Val Loss: 0.4729 | Val Recall: 0.0052 | Val Precision: 0.0844 |Val F1: 0.0099 |Val Accuracy: 0.8783
Epoch 4/10 | Train Loss: 0.4554 | Val Loss: 0.3934 | Val Recall: 0.0917 | Val Precision: 0.2688 |Val F1: 0.1368 |Val Accuracy: 0.9011
Epoch 5/10 | Train Loss: 0.3797 | Val Loss: 0.3420 | Val Recall: 0.2318 | Val Precision: 0.5630 |Val F1: 0.3283 |Val Accuracy: 0.9207
Epoch 6/10 | Train Loss: 0.3275 | Val Loss: 0.3027 | Val Recall: 0.3277 | Val Precision: 0.6458 |Val F1: 0.4347 |Val Accuracy: 0.9329
Epoch 7/10 | Train Loss: 0.2897 | Val Loss: 0.2774 | Val Recall: 0.3609 | Val Precision: 0.6903 |Val F1: 0.4740 |Val Accuracy: 0.9361
Epoch 8/10 | Train Loss: 0.2613 | Val Loss: 0.2577 | Val Recall: 0.3999 | Val Precision: 0.7239 |Val F1: 0.5152 |Val Accuracy: 0.9390
Epoch 9/10 | Train Loss: 0.2403 | Val Loss: 0.2382 | Val Recall: 0.4806 | Val Precision: 0.7586 |Val F1: 0.5884 |Val Accuracy: 0.9448
Epoch 10/10 | Train Loss: 0.2260 | Val Loss: 0.2251 | Val Recall: 0.5358 | Val Precision: 0.7785 |Val F1: 0.6347 |Val Accuracy: 0.9482
```

```
Training complete!
```

```
# plot evaluation metrics
plot_training_metrics(history_v1)
```

```
/var/folders/qw/2jrbyvsn4tnckwl2l1rgl1wm0000gp/T/ipykernel_54182/4064934705.py:20: UserWarning: linestyle is redundantly defined by the 'linest
  plt.plot(epochs_range, history['accuracy'], 'k-x', label='Accuracy', linestyle='--')
```



Observation: Model is actually learning and the overfitting at epoch 10 is almost negligible. But, a high accuracy of 94.82% with a low recall and precision of 53.58% shows that the model has learnt that predicting non-entity (label 'O') most of the times is a safe bet.

Implementing weighted loss strategy to tell the model that missing an entity is worst than getting a non-entity wrong.

To calculate weights, if the tag occurs more number of times like the non-entity tag 'O', we need to give it lower weight. Using sklearn class_weight utility to compute this.

```
from sklearn.utils.class_weight import compute_class_weight
#list of all tags in training set
all_classes = [label
              for row in train_distilbert['labels']
                for label in row
                  if label != -100
]
unique_classes = np.unique(all_classes)

#balanced mode adjusts the weights inversly proportional to the frequencies of the classes
weights = compute_class_weight(class_weight='balanced', classes = unique_classes, y = all_classes)

#convert class weights to pytorch tensor
class_weights = torch.tensor(weights, dtype=torch.float).to(device)

print("Calculated Class Weights:")
for i, weight in enumerate(class_weights):
    # getting label names from id2label
    label_name = id2label[i] if 'id2label' in locals() else str(i)
    print(f"{label_name}: {weight:.4f}")
```

```
Calculated Class Weights:
B-CODE: 15.6851
B-DATETIME: 2.4340
B-DEM: 33.6022
B-LOC: 13.7717
B-MISC: 68.8216
B-ORG: 9.5366
B-PERSON: 5.0459
B-QUANTITY: 45.8811
I-CODE: 504.6920
I-DATETIME: 1.4053
```

```
I-DEM: 54.5324
I-LOC: 41.7168
I-MISC: 19.3093
I-ORG: 4.0306
I-PERSON: 2.7905
I-QUANTITY: 30.2104
O: 0.0683
```

```
#trying similar architecture with weighted loss approach
lstm_nre_v2 = bilstm_nre(30522, len(label_list), embed_dim=128, hidden_dim=256, weight_tensor=class_weights).to(device)
optimizer_v2 = torch.optim.Adam(lstm_nre_v2.parameters(), lr=1e-4)
```

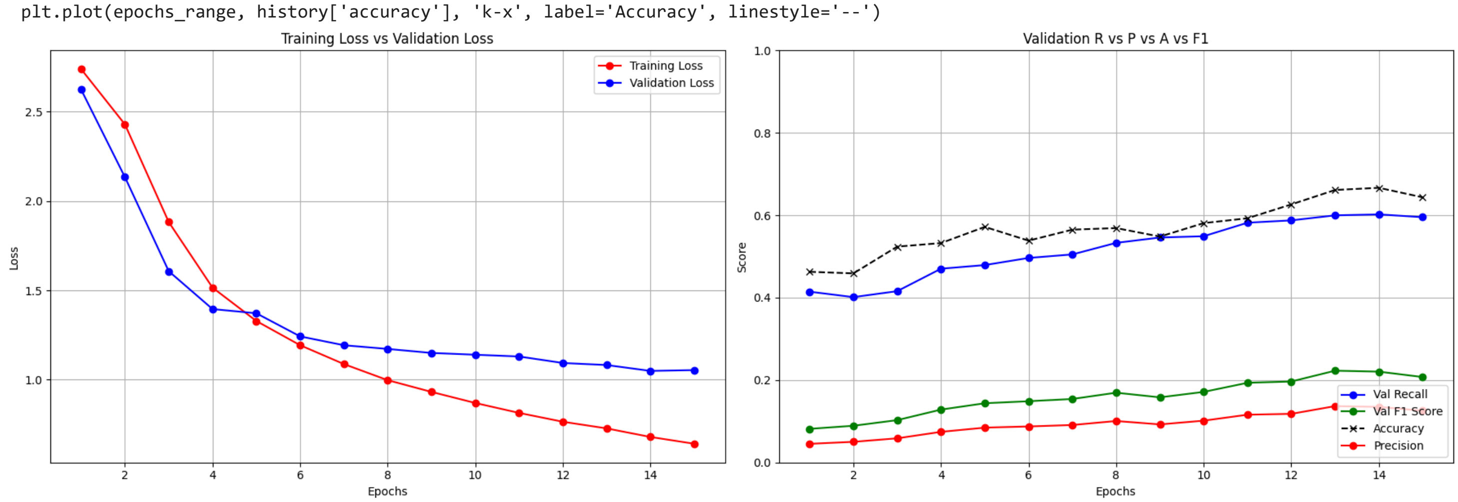
Will the model learn more if I increase the number of epoches?

```
history_v2 = train_eval_lstm(lstm_nre_v2, optimizer_v2, n_epoches = 15)
```

```
Starting Bi-LSTM for NER training...
Epoch 1/15 | Train Loss: 2.7368 | Val Loss: 2.6221 | Val Recall: 0.4145 | Val Precision: 0.0453 |Val F1: 0.0816 |Val Accuracy: 0.4628
Epoch 2/15 | Train Loss: 2.4277 | Val Loss: 2.1327 | Val Recall: 0.4013 | Val Precision: 0.0501 |Val F1: 0.0891 |Val Accuracy: 0.4589
Epoch 3/15 | Train Loss: 1.8818 | Val Loss: 1.6070 | Val Recall: 0.4157 | Val Precision: 0.0586 |Val F1: 0.1028 |Val Accuracy: 0.5238
Epoch 4/15 | Train Loss: 1.5152 | Val Loss: 1.3953 | Val Recall: 0.4703 | Val Precision: 0.0743 |Val F1: 0.1283 |Val Accuracy: 0.5324
Epoch 5/15 | Train Loss: 1.3286 | Val Loss: 1.3718 | Val Recall: 0.4790 | Val Precision: 0.0845 |Val F1: 0.1437 |Val Accuracy: 0.5717
Epoch 6/15 | Train Loss: 1.1932 | Val Loss: 1.2425 | Val Recall: 0.4964 | Val Precision: 0.0874 |Val F1: 0.1486 |Val Accuracy: 0.5383
Epoch 7/15 | Train Loss: 1.0877 | Val Loss: 1.1930 | Val Recall: 0.5049 | Val Precision: 0.0909 |Val F1: 0.1541 |Val Accuracy: 0.5650
Epoch 8/15 | Train Loss: 0.9977 | Val Loss: 1.1724 | Val Recall: 0.5329 | Val Precision: 0.1007 |Val F1: 0.1693 |Val Accuracy: 0.5688
Epoch 9/15 | Train Loss: 0.9321 | Val Loss: 1.1499 | Val Recall: 0.5459 | Val Precision: 0.0924 |Val F1: 0.1580 |Val Accuracy: 0.5482
Epoch 10/15 | Train Loss: 0.8701 | Val Loss: 1.1399 | Val Recall: 0.5491 | Val Precision: 0.1015 |Val F1: 0.1713 |Val Accuracy: 0.5809
Epoch 11/15 | Train Loss: 0.8146 | Val Loss: 1.1301 | Val Recall: 0.5819 | Val Precision: 0.1159 |Val F1: 0.1933 |Val Accuracy: 0.5927
Epoch 12/15 | Train Loss: 0.7652 | Val Loss: 1.0938 | Val Recall: 0.5876 | Val Precision: 0.1180 |Val F1: 0.1965 |Val Accuracy: 0.6263
Epoch 13/15 | Train Loss: 0.7280 | Val Loss: 1.0823 | Val Recall: 0.5997 | Val Precision: 0.1368 |Val F1: 0.2227 |Val Accuracy: 0.6613
Epoch 14/15 | Train Loss: 0.6801 | Val Loss: 1.0495 | Val Recall: 0.6020 | Val Precision: 0.1349 |Val F1: 0.2204 |Val Accuracy: 0.6666
Epoch 15/15 | Train Loss: 0.6422 | Val Loss: 1.0541 | Val Recall: 0.5953 | Val Precision: 0.1255 |Val F1: 0.2073 |Val Accuracy: 0.6435
Training complete!
```

```
# plot evaluation metrics
plot_training_metrics(history_v2)
```

```
/var/folders/qw/2jrbyvsn4tnckwl2l1rgl1wm0000gp/T/ipykernel_54182/4064934705.py:20: UserWarning: linestyle is redundantly defined by the 'linest
  plt.plot(epochs_range, history['accuracy'], 'k-x', label='Accuracy', linestyle='--')
```



Observed that the model starts to overfit at epoch 5. The recall at epoch 14 has improved to 60.20% the precision has drastically dropped to 13.49%, showing that only 13.49% of the redacted words are actually sensetive! The accuracy has also dropped to 64.35%. The fact that the difference between the recall and accuracy has dropped is good.

What effect might it have if the number of nodes in the hidden layer of the network is increased while sticking to weighted loss strategy?

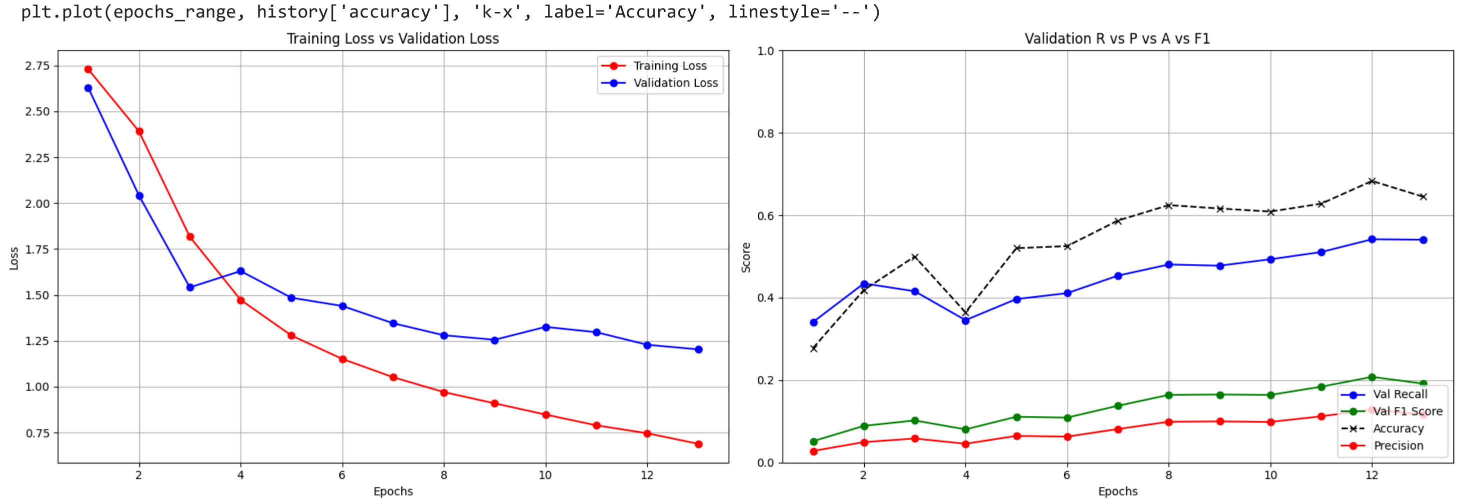
```
#increasing dimension of hidden layer without using weighted loss
lstm_nre_v3 = bilstm_nre(30522, len(label_list), embed_dim=128, hidden_dim=288, weight_tensor=class_weights).to(device)
optimizer_v3 = torch.optim.Adam(lstm_nre_v3.parameters(), lr=1e-4)
```

```
history_v3 = train_eval_lstm(lstm_nre_v3, optimizer_v3, n_epoches = 13)
```

```
Starting Bi-LSTM for NER training...
Epoch 1/13 | Train Loss: 2.7300 | Val Loss: 2.6301 | Val Recall: 0.3408 | Val Precision: 0.0280 |Val F1: 0.0517 |Val Accuracy: 0.2777
Epoch 2/13 | Train Loss: 2.3913 | Val Loss: 2.0419 | Val Recall: 0.4345 | Val Precision: 0.0495 |Val F1: 0.0888 |Val Accuracy: 0.4185
Epoch 3/13 | Train Loss: 1.8169 | Val Loss: 1.5410 | Val Recall: 0.4156 | Val Precision: 0.0581 |Val F1: 0.1020 |Val Accuracy: 0.4998
Epoch 4/13 | Train Loss: 1.4734 | Val Loss: 1.6300 | Val Recall: 0.3454 | Val Precision: 0.0454 |Val F1: 0.0803 |Val Accuracy: 0.3642
Epoch 5/13 | Train Loss: 1.2789 | Val Loss: 1.4846 | Val Recall: 0.3966 | Val Precision: 0.0645 |Val F1: 0.1110 |Val Accuracy: 0.5201
Epoch 6/13 | Train Loss: 1.1515 | Val Loss: 1.4398 | Val Recall: 0.4109 | Val Precision: 0.0628 |Val F1: 0.1089 |Val Accuracy: 0.5251
Epoch 7/13 | Train Loss: 1.0518 | Val Loss: 1.3458 | Val Recall: 0.4535 | Val Precision: 0.0812 |Val F1: 0.1377 |Val Accuracy: 0.5872
Epoch 8/13 | Train Loss: 0.9707 | Val Loss: 1.2802 | Val Recall: 0.4805 | Val Precision: 0.0989 |Val F1: 0.1640 |Val Accuracy: 0.6246
Epoch 9/13 | Train Loss: 0.9093 | Val Loss: 1.2557 | Val Recall: 0.4776 | Val Precision: 0.0997 |Val F1: 0.1650 |Val Accuracy: 0.6163
Epoch 10/13 | Train Loss: 0.8484 | Val Loss: 1.3259 | Val Recall: 0.4932 | Val Precision: 0.0984 |Val F1: 0.1640 |Val Accuracy: 0.6089
Epoch 11/13 | Train Loss: 0.7896 | Val Loss: 1.2968 | Val Recall: 0.5107 | Val Precision: 0.1121 |Val F1: 0.1838 |Val Accuracy: 0.6281
Epoch 12/13 | Train Loss: 0.7465 | Val Loss: 1.2286 | Val Recall: 0.5417 | Val Precision: 0.1285 |Val F1: 0.2077 |Val Accuracy: 0.6832
Epoch 13/13 | Train Loss: 0.6892 | Val Loss: 1.2035 | Val Recall: 0.5406 | Val Precision: 0.1163 |Val F1: 0.1914 |Val Accuracy: 0.6451
Training complete!
```

```
# plot evaluation metrics
plot_training_metrics(history_v3)
```

```
/var/folders/qw/2jrbyvsn4tnckwl2l1rgl1wm0000gp/T/ipykernel_54182/4064934705.py:20: UserWarning: linestyle is redundantly defined by the 'linest
  plt.plot(epochs_range, history['accuracy'], 'k-x', label='Accuracy', linestyle='--')
```



Observation: The model started to overfit at epoch 4. It kept learning however, the overfittin kept increasing. The best recall of 54.17% was achieved at epoch 12 which is less than the previous. Also, precision dropped to 12.85%.

Trying to decrease the number of nodes in the hidden layer.

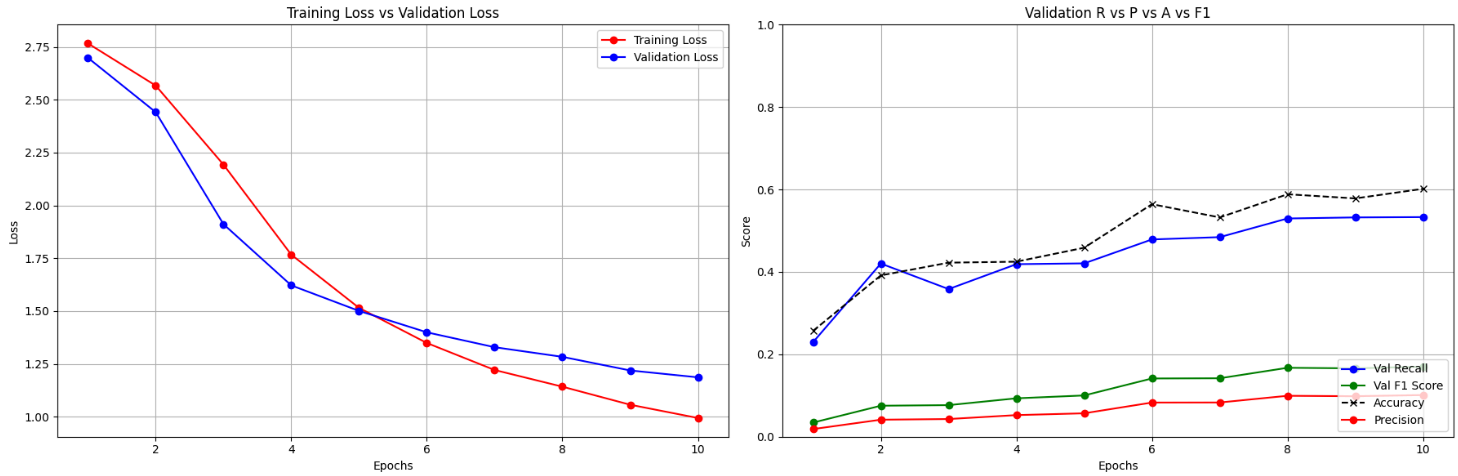
```
#decreasing dimension of hidden layer without using weighted loss
lstm_nre_v4 = bilstm_nre(30522, len(label_list), embed_dim=128, hidden_dim=192, weight_tensor=class_weights).to(device)
optimizer_v4 = torch.optim.Adam(lstm_nre_v4.parameters(), lr=1e-4)
```

```
history_v4 = train_eval_lstm(lstm_nre_v4, optimizer_v4, n_epoches = 10)
```

```
Starting Bi-LSTM for NER training...
Epoch 1/10 | Train Loss: 2.7665 | Val Loss: 2.6983 | Val Recall: 0.2294 | Val Precision: 0.0186 |Val F1: 0.0343 |Val Accuracy: 0.2569
Epoch 2/10 | Train Loss: 2.5669 | Val Loss: 2.4411 | Val Recall: 0.4203 | Val Precision: 0.0413 |Val F1: 0.0752 |Val Accuracy: 0.3915
Epoch 3/10 | Train Loss: 2.1927 | Val Loss: 1.9109 | Val Recall: 0.3583 | Val Precision: 0.0429 |Val F1: 0.0767 |Val Accuracy: 0.4224
Epoch 4/10 | Train Loss: 1.7667 | Val Loss: 1.6215 | Val Recall: 0.4188 | Val Precision: 0.0525 |Val F1: 0.0932 |Val Accuracy: 0.4248
Epoch 5/10 | Train Loss: 1.5149 | Val Loss: 1.5006 | Val Recall: 0.4207 | Val Precision: 0.0569 |Val F1: 0.1002 |Val Accuracy: 0.4587
Epoch 6/10 | Train Loss: 1.3489 | Val Loss: 1.3993 | Val Recall: 0.4788 | Val Precision: 0.0829 |Val F1: 0.1414 |Val Accuracy: 0.5643
Epoch 7/10 | Train Loss: 1.2209 | Val Loss: 1.3288 | Val Recall: 0.4844 | Val Precision: 0.0831 |Val F1: 0.1419 |Val Accuracy: 0.5322
Epoch 8/10 | Train Loss: 1.1420 | Val Loss: 1.2833 | Val Recall: 0.5297 | Val Precision: 0.0994 |Val F1: 0.1673 |Val Accuracy: 0.5884
Epoch 9/10 | Train Loss: 1.0566 | Val Loss: 1.2188 | Val Recall: 0.5322 | Val Precision: 0.0983 |Val F1: 0.1660 |Val Accuracy: 0.5783
Epoch 10/10 | Train Loss: 0.9938 | Val Loss: 1.1863 | Val Recall: 0.5330 | Val Precision: 0.1011 |Val F1: 0.1700 |Val Accuracy: 0.6017
Training complete!
```

```
# plot evaluation metrics
plot_training_metrics(history_v4)
```

```
/var/folders/qw/2jrbyvsn4tnckwl2l1rgl1wm0000gp/T/ipykernel_54182/4064934705.py:20: UserWarning: linestyle is redundantly defined by the 'linest
  plt.plot(epochs_range, history['accuracy'], 'k-x', label='Accuracy', linestyle='--')
```



Observation: The best recall for this strategy is still 53.30% with precision of 10.11%. Going back to v2 and trying to recude overfitting by adding dropout.

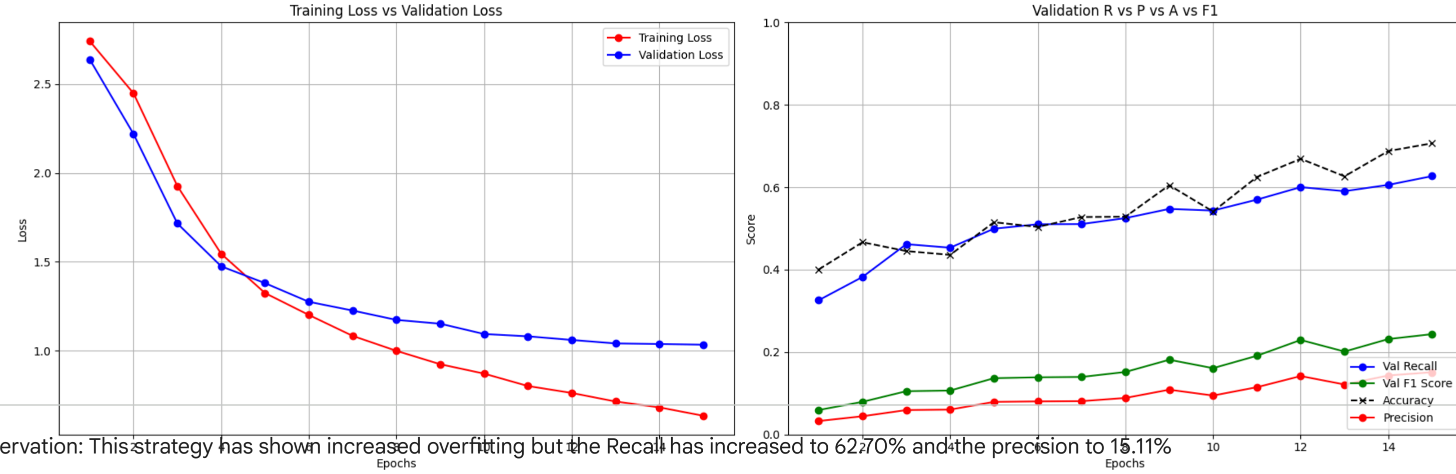
```
#trying similar architecture with weighted loss approach
lstm_nre_v5 = bilstm_nre(30522, len(label_list), embed_dim=128, hidden_dim=256, weight_tensor=class_weights, dropout_rate=0.2).to(device)
optimizer_v5 = torch.optim.Adam(lstm_nre_v5.parameters(), lr=1e-4)
```

```
history_v5 = train_eval_lstm(lstm_nre_v5, optimizer_v5, n_epoches = 15)
```

```
Starting Bi-LSTM for NER training...
Epoch 1/15 | Train Loss: 2.7417 | Val Loss: 2.6390 | Val Recall: 0.3258 | Val Precision: 0.0327 |Val F1: 0.0594 |Val Accuracy: 0.4004
Epoch 2/15 | Train Loss: 2.4480 | Val Loss: 2.2191 | Val Recall: 0.3823 | Val Precision: 0.0442 |Val F1: 0.0792 |Val Accuracy: 0.4665
Epoch 3/15 | Train Loss: 1.9265 | Val Loss: 1.7158 | Val Recall: 0.4618 | Val Precision: 0.0592 |Val F1: 0.1050 |Val Accuracy: 0.4450
Epoch 4/15 | Train Loss: 1.5451 | Val Loss: 1.4748 | Val Recall: 0.4533 | Val Precision: 0.0604 |Val F1: 0.1066 |Val Accuracy: 0.4358
Epoch 5/15 | Train Loss: 1.3244 | Val Loss: 1.3801 | Val Recall: 0.4994 | Val Precision: 0.0791 |Val F1: 0.1366 |Val Accuracy: 0.5151
Epoch 6/15 | Train Loss: 1.2011 | Val Loss: 1.2751 | Val Recall: 0.5101 | Val Precision: 0.0803 |Val F1: 0.1387 |Val Accuracy: 0.5037
Epoch 7/15 | Train Loss: 1.0833 | Val Loss: 1.2259 | Val Recall: 0.5107 | Val Precision: 0.0809 |Val F1: 0.1396 |Val Accuracy: 0.5277
Epoch 8/15 | Train Loss: 0.9991 | Val Loss: 1.1732 | Val Recall: 0.5250 | Val Precision: 0.0886 |Val F1: 0.1517 |Val Accuracy: 0.5284
Epoch 9/15 | Train Loss: 0.9238 | Val Loss: 1.1519 | Val Recall: 0.5474 | Val Precision: 0.1087 |Val F1: 0.1813 |Val Accuracy: 0.6049
Epoch 10/15 | Train Loss: 0.8710 | Val Loss: 1.0938 | Val Recall: 0.5433 | Val Precision: 0.0945 |Val F1: 0.1609 |Val Accuracy: 0.5410
Epoch 11/15 | Train Loss: 0.8013 | Val Loss: 1.0809 | Val Recall: 0.5699 | Val Precision: 0.1147 |Val F1: 0.1910 |Val Accuracy: 0.6248
Epoch 12/15 | Train Loss: 0.7622 | Val Loss: 1.0604 | Val Recall: 0.6002 | Val Precision: 0.1418 |Val F1: 0.2293 |Val Accuracy: 0.6690
Epoch 13/15 | Train Loss: 0.7140 | Val Loss: 1.0409 | Val Recall: 0.5904 | Val Precision: 0.1213 |Val F1: 0.2012 |Val Accuracy: 0.6265
Epoch 14/15 | Train Loss: 0.6811 | Val Loss: 1.0377 | Val Recall: 0.6058 | Val Precision: 0.1431 |Val F1: 0.2316 |Val Accuracy: 0.6878
Epoch 15/15 | Train Loss: 0.6340 | Val Loss: 1.0339 | Val Recall: 0.6270 | Val Precision: 0.1511 |Val F1: 0.2435 |Val Accuracy: 0.7067
Training complete!
```

```
# plot evaluation metrics
plot_training_metrics(history_v5)
```

```
/var/folders/qw/2jrbyvsn4tnckwl2l1rgl1wm0000gp/T/ipykernel_54182/4064934705.py:20: UserWarning: linestyle is redundantly defined by the 'linest
  plt.plot(epochs_range, history['accuracy'], 'k-x', label='Accuracy', linestyle='--')
```



Observation: This strategy has shown increased overfitting but the Recall has increased to 62.70% and the precision to 15.11%

Trying an approach where increasing the number of nodes in the hidden layer without the weighted loss or dropout strategy.

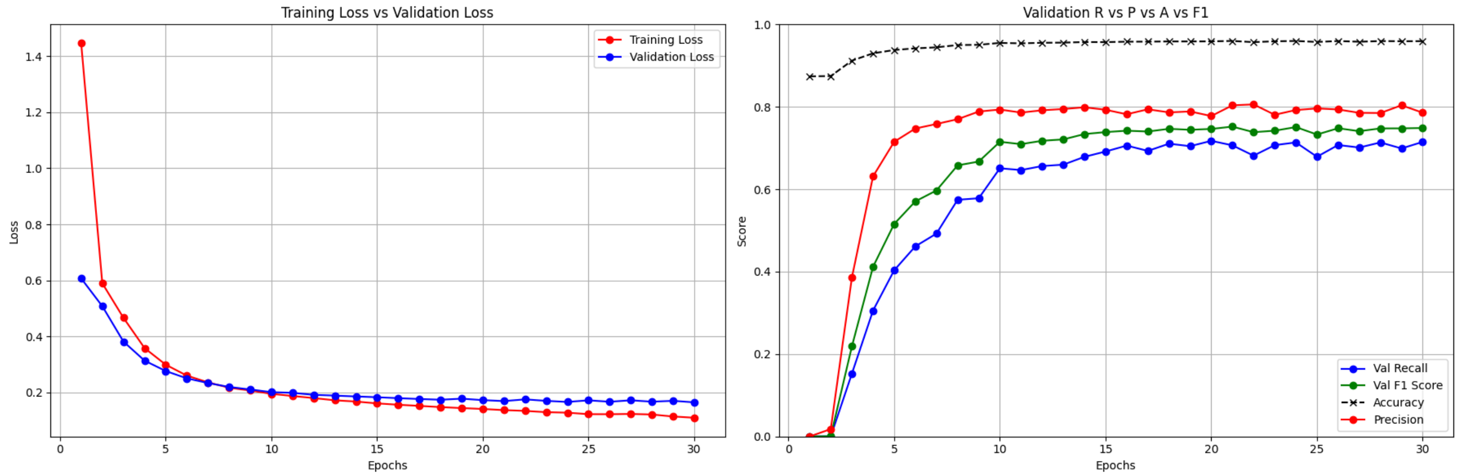
```
#increasing dimension of hidden layer more without using weighted loss
lstm_nre_v6 = bilstm_nre(30522, len(label_list), embed_dim=128, hidden_dim=480).to(device)
optimizer_v6 = torch.optim.Adam(lstm_nre_v6.parameters(), lr=1e-4)
```

```
history_v6 = train_eval_lstm(lstm_nre_v6, optimizer_v6, n_epoches = 30)
```

```
Starting Bi-LSTM for NER training...
/opt/anaconda3/envs/hf-venv/lib/python3.11/site-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-define
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/hf-venv/lib/python3.11/site-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-define
  _warn_prf(average, modifier, msg_start, len(result))
Epoch 1/30 | Train Loss: 1.4469 | Val Loss: 0.6069 | Val Recall: 0.0000 | Val Precision: 0.0000 |Val F1: 0.0000 |Val Accuracy: 0.8740
Epoch 2/30 | Train Loss: 0.5894 | Val Loss: 0.5076 | Val Recall: 0.0003 | Val Precision: 0.0180 |Val F1: 0.0005 |Val Accuracy: 0.8747
Epoch 3/30 | Train Loss: 0.4662 | Val Loss: 0.3813 | Val Recall: 0.1525 | Val Precision: 0.3858 |Val F1: 0.2186 |Val Accuracy: 0.9120
Epoch 4/30 | Train Loss: 0.3580 | Val Loss: 0.3131 | Val Recall: 0.3056 | Val Precision: 0.6313 |Val F1: 0.4119 |Val Accuracy: 0.9299
Epoch 5/30 | Train Loss: 0.2987 | Val Loss: 0.2761 | Val Recall: 0.4033 | Val Precision: 0.7152 |Val F1: 0.5158 |Val Accuracy: 0.9378
Epoch 6/30 | Train Loss: 0.2596 | Val Loss: 0.2498 | Val Recall: 0.4614 | Val Precision: 0.7473 |Val F1: 0.5705 |Val Accuracy: 0.9419
Epoch 7/30 | Train Loss: 0.2355 | Val Loss: 0.2332 | Val Recall: 0.4924 | Val Precision: 0.7583 |Val F1: 0.5971 |Val Accuracy: 0.9444
Epoch 8/30 | Train Loss: 0.2164 | Val Loss: 0.2191 | Val Recall: 0.5744 | Val Precision: 0.7699 |Val F1: 0.6580 |Val Accuracy: 0.9499
Epoch 9/30 | Train Loss: 0.2065 | Val Loss: 0.2102 | Val Recall: 0.5781 | Val Precision: 0.7885 |Val F1: 0.6671 |Val Accuracy: 0.9505
Epoch 10/30 | Train Loss: 0.1945 | Val Loss: 0.2006 | Val Recall: 0.6509 | Val Precision: 0.7935 |Val F1: 0.7151 |Val Accuracy: 0.9552
Epoch 11/30 | Train Loss: 0.1869 | Val Loss: 0.1983 | Val Recall: 0.6465 | Val Precision: 0.7860 |Val F1: 0.7095 |Val Accuracy: 0.9541
Epoch 12/30 | Train Loss: 0.1794 | Val Loss: 0.1910 | Val Recall: 0.6563 | Val Precision: 0.7916 |Val F1: 0.7176 |Val Accuracy: 0.9555
Epoch 13/30 | Train Loss: 0.1715 | Val Loss: 0.1884 | Val Recall: 0.6597 | Val Precision: 0.7947 |Val F1: 0.7209 |Val Accuracy: 0.9558
Epoch 14/30 | Train Loss: 0.1671 | Val Loss: 0.1852 | Val Recall: 0.6787 | Val Precision: 0.7989 |Val F1: 0.7339 |Val Accuracy: 0.9572
Epoch 15/30 | Train Loss: 0.1604 | Val Loss: 0.1825 | Val Recall: 0.6915 | Val Precision: 0.7928 |Val F1: 0.7387 |Val Accuracy: 0.9570
Epoch 16/30 | Train Loss: 0.1556 | Val Loss: 0.1792 | Val Recall: 0.7059 | Val Precision: 0.7819 |Val F1: 0.7420 |Val Accuracy: 0.9581
Epoch 17/30 | Train Loss: 0.1515 | Val Loss: 0.1762 | Val Recall: 0.6930 | Val Precision: 0.7940 |Val F1: 0.7401 |Val Accuracy: 0.9582
Epoch 18/30 | Train Loss: 0.1473 | Val Loss: 0.1735 | Val Recall: 0.7105 | Val Precision: 0.7865 |Val F1: 0.7466 |Val Accuracy: 0.9587
Epoch 19/30 | Train Loss: 0.1437 | Val Loss: 0.1772 | Val Recall: 0.7045 | Val Precision: 0.7887 |Val F1: 0.7442 |Val Accuracy: 0.9588
Epoch 20/30 | Train Loss: 0.1406 | Val Loss: 0.1721 | Val Recall: 0.7175 | Val Precision: 0.7777 |Val F1: 0.7464 |Val Accuracy: 0.9589
Epoch 21/30 | Train Loss: 0.1364 | Val Loss: 0.1687 | Val Recall: 0.7065 | Val Precision: 0.8037 |Val F1: 0.7520 |Val Accuracy: 0.9599
Epoch 22/30 | Train Loss: 0.1337 | Val Loss: 0.1749 | Val Recall: 0.6815 | Val Precision: 0.8059 |Val F1: 0.7385 |Val Accuracy: 0.9569
Epoch 23/30 | Train Loss: 0.1291 | Val Loss: 0.1695 | Val Recall: 0.7070 | Val Precision: 0.7807 |Val F1: 0.7420 |Val Accuracy: 0.9591
Epoch 24/30 | Train Loss: 0.1273 | Val Loss: 0.1656 | Val Recall: 0.7136 | Val Precision: 0.7922 |Val F1: 0.7509 |Val Accuracy: 0.9599
Epoch 25/30 | Train Loss: 0.1220 | Val Loss: 0.1714 | Val Recall: 0.6788 | Val Precision: 0.7961 |Val F1: 0.7328 |Val Accuracy: 0.9575
Epoch 26/30 | Train Loss: 0.1218 | Val Loss: 0.1663 | Val Recall: 0.7073 | Val Precision: 0.7936 |Val F1: 0.7480 |Val Accuracy: 0.9597
Epoch 27/30 | Train Loss: 0.1228 | Val Loss: 0.1716 | Val Recall: 0.7012 | Val Precision: 0.7855 |Val F1: 0.7409 |Val Accuracy: 0.9580
Epoch 28/30 | Train Loss: 0.1208 | Val Loss: 0.1665 | Val Recall: 0.7135 | Val Precision: 0.7850 |Val F1: 0.7476 |Val Accuracy: 0.9596
Epoch 29/30 | Train Loss: 0.1137 | Val Loss: 0.1696 | Val Recall: 0.6990 | Val Precision: 0.8038 |Val F1: 0.7477 |Val Accuracy: 0.9592
Epoch 30/30 | Train Loss: 0.1092 | Val Loss: 0.1642 | Val Recall: 0.7149 | Val Precision: 0.7858 |Val F1: 0.7487 |Val Accuracy: 0.9594
Training complete!
```

```
# plot evaluation metrics
plot_training_metrics(history_v6)
```

```
/var/folders/qw/2jrbyvsn4tnckwl2l1rgl1wm0000gp/T/ipykernel_54182/4064934705.py:20: UserWarning: linestyle is redundantly defined by the 'linest
  plt.plot(epochs_range, history['accuracy'], 'k-x', label='Accuracy', linestyle='--')
```



Observation: After epoch 10 the model is learning slowly but could reach 71.75% recall and 77.77% precision at epoch 20 however, the model is not learning much after that. Some amount of overfitting is seen after epoch 10.

Adding dropout layers to reduce overfitting.

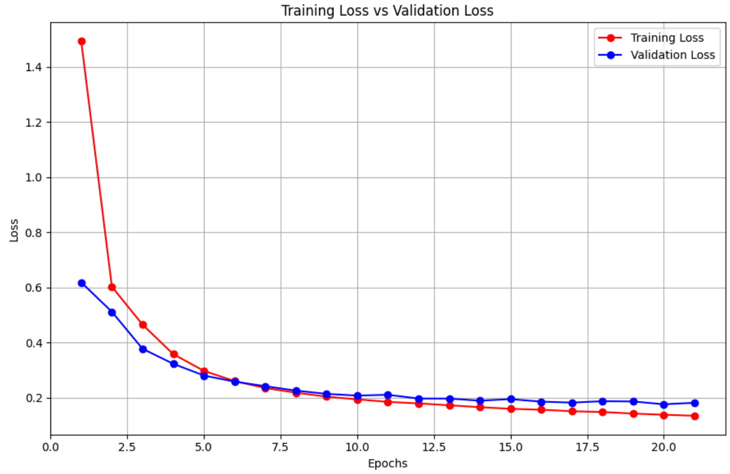
```
#adding dropout layer to reduce overfitting and reducing epoches
lstm_nre_v7 = bilstm_nre(30522, len(label_list), embed_dim=128, hidden_dim=480, dropout_rate=0.2).to(device)
optimizer_v7 = torch.optim.Adam(lstm_nre_v7.parameters(), lr=1e-4)
```

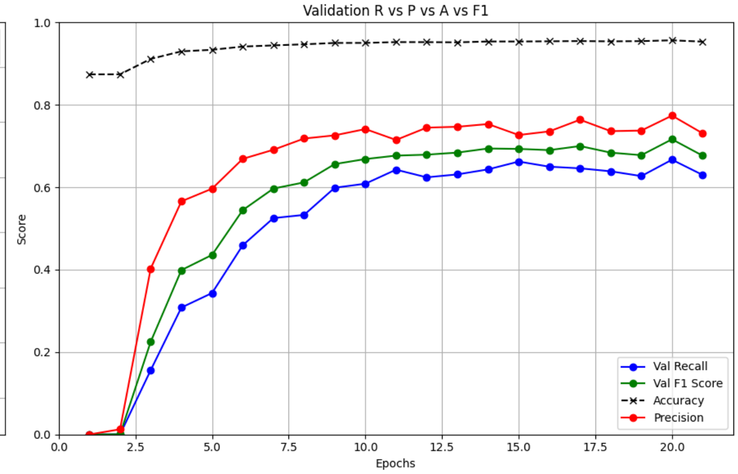
```
history_v7 = train_eval_lstm(lstm_nre_v7, optimizer_v7, n_epoches = 21)
```

```
Starting Bi-LSTM for NER training...
/opt/anaconda3/envs/hf-venv/lib/python3.11/site-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-define
  _warn_prf(average, modifier, msg_start, len(result))
/opt/anaconda3/envs/hf-venv/lib/python3.11/site-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-define
  _warn_prf(average, modifier, msg_start, len(result))
Epoch 1/21 | Train Loss: 1.4933 | Val Loss: 0.6183 | Val Recall: 0.0000 | Val Precision: 0.0000 |Val F1: 0.0000 |Val Accuracy: 0.8740
Epoch 2/21 | Train Loss: 0.6024 | Val Loss: 0.5115 | Val Recall: 0.0003 | Val Precision: 0.0126 |Val F1: 0.0005 |Val Accuracy: 0.8741
Epoch 3/21 | Train Loss: 0.4649 | Val Loss: 0.3773 | Val Recall: 0.1562 | Val Precision: 0.4029 |Val F1: 0.2251 |Val Accuracy: 0.9114
Epoch 4/21 | Train Loss: 0.3578 | Val Loss: 0.3229 | Val Recall: 0.3080 | Val Precision: 0.5658 |Val F1: 0.3989 |Val Accuracy: 0.9298
Epoch 5/21 | Train Loss: 0.2973 | Val Loss: 0.2803 | Val Recall: 0.3433 | Val Precision: 0.5965 |Val F1: 0.4358 |Val Accuracy: 0.9335
Epoch 6/21 | Train Loss: 0.2602 | Val Loss: 0.2581 | Val Recall: 0.4591 | Val Precision: 0.6688 |Val F1: 0.5444 |Val Accuracy: 0.9414
Epoch 7/21 | Train Loss: 0.2349 | Val Loss: 0.2417 | Val Recall: 0.5251 | Val Precision: 0.6909 |Val F1: 0.5967 |Val Accuracy: 0.9442
Epoch 8/21 | Train Loss: 0.2181 | Val Loss: 0.2255 | Val Recall: 0.5329 | Val Precision: 0.7182 |Val F1: 0.6118 |Val Accuracy: 0.9469
Epoch 9/21 | Train Loss: 0.2035 | Val Loss: 0.2137 | Val Recall: 0.5987 | Val Precision: 0.7259 |Val F1: 0.6562 |Val Accuracy: 0.9502
Epoch 10/21 | Train Loss: 0.1939 | Val Loss: 0.2074 | Val Recall: 0.6086 | Val Precision: 0.7412 |Val F1: 0.6684 |Val Accuracy: 0.9503
Epoch 11/21 | Train Loss: 0.1847 | Val Loss: 0.2104 | Val Recall: 0.6426 | Val Precision: 0.7148 |Val F1: 0.6768 |Val Accuracy: 0.9522
Epoch 12/21 | Train Loss: 0.1791 | Val Loss: 0.1968 | Val Recall: 0.6240 | Val Precision: 0.7446 |Val F1: 0.6790 |Val Accuracy: 0.9522
Epoch 13/21 | Train Loss: 0.1723 | Val Loss: 0.1966 | Val Recall: 0.6311 | Val Precision: 0.7467 |Val F1: 0.6841 |Val Accuracy: 0.9517
Epoch 14/21 | Train Loss: 0.1654 | Val Loss: 0.1891 | Val Recall: 0.6433 | Val Precision: 0.7534 |Val F1: 0.6940 |Val Accuracy: 0.9537
Epoch 15/21 | Train Loss: 0.1595 | Val Loss: 0.1945 | Val Recall: 0.6623 | Val Precision: 0.7267 |Val F1: 0.6930 |Val Accuracy: 0.9537
Epoch 16/21 | Train Loss: 0.1562 | Val Loss: 0.1856 | Val Recall: 0.6499 | Val Precision: 0.7356 |Val F1: 0.6901 |Val Accuracy: 0.9542
Epoch 17/21 | Train Loss: 0.1507 | Val Loss: 0.1820 | Val Recall: 0.6459 | Val Precision: 0.7638 |Val F1: 0.6999 |Val Accuracy: 0.9548
Epoch 18/21 | Train Loss: 0.1478 | Val Loss: 0.1871 | Val Recall: 0.6387 | Val Precision: 0.7362 |Val F1: 0.6840 |Val Accuracy: 0.9540
Epoch 19/21 | Train Loss: 0.1422 | Val Loss: 0.1862 | Val Recall: 0.6271 | Val Precision: 0.7375 |Val F1: 0.6778 |Val Accuracy: 0.9546
Epoch 20/21 | Train Loss: 0.1379 | Val Loss: 0.1758 | Val Recall: 0.6668 | Val Precision: 0.7738 |Val F1: 0.7164 |Val Accuracy: 0.9566
Epoch 21/21 | Train Loss: 0.1343 | Val Loss: 0.1814 | Val Recall: 0.6298 | Val Precision: 0.7308 |Val F1: 0.6766 |Val Accuracy: 0.9534
Training complete!
```

```
# plot evaluation metrics
plot_training_metrics(history_v7)
```

```
/var/folders/qw/2jrbyvsn4tnckwl2l1rgl1wm0000gp/T/ipykernel_54182/4064934705.py:20: UserWarning: linestyle is redundantly defined by the 'linest
  plt.plot(epochs_range, history['accuracy'], 'k-x', label='Accuracy', linestyle='--')
```





Observation: There is not much change in the overfitting but the max recall achieved at epoch 20 is 66.68% which is lower than the previous model.

Implementing early stopping on v6 for final training run.

```
#increasing dimension of hidden layer more without using weighted loss
lstm_nre_v8 = bilstm_nre(30522, len(label_list), embed_dim=128, hidden_dim=480).to(device)
optimizer_v8 = torch.optim.Adam(lstm_nre_v8.parameters(), lr=1e-4)
```

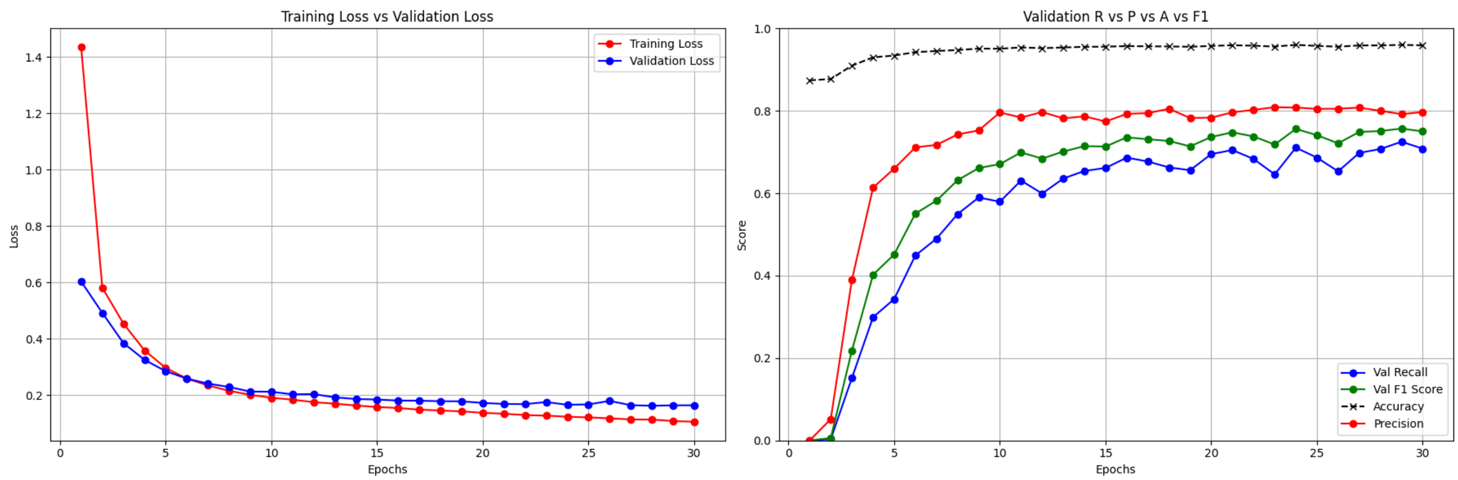
```
history_v8 = train_eval_lstm(lstm_nre_v8, optimizer_v8, n_epoches = 30, early_stopping=True)
```

```
Starting Bi-LSTM for NER training...
Epoch 1/30 | Train Loss: 1.4340 | Val Loss: 0.6030 | Val Recall: 0.0000 | Val Precision: 0.0000 |Val F1: 0.0000 |Val Accuracy: 0.8740
Validation Loss improved from inf to 0.6030.
Epoch 2/30 | Train Loss: 0.5813 | Val Loss: 0.4920 | Val Recall: 0.0031 | Val Precision: 0.0518 |Val F1: 0.0059 |Val Accuracy: 0.8771
Validation Loss improved from 0.6030 to 0.4920.
Epoch 3/30 | Train Loss: 0.4541 | Val Loss: 0.3848 | Val Recall: 0.1516 | Val Precision: 0.3897 |Val F1: 0.2183 |Val Accuracy: 0.9097
Validation Loss improved from 0.4920 to 0.3848.
Epoch 4/30 | Train Loss: 0.3587 | Val Loss: 0.3252 | Val Recall: 0.2991 | Val Precision: 0.6135 |Val F1: 0.4021 |Val Accuracy: 0.9298
Validation Loss improved from 0.3848 to 0.3252.
Epoch 5/30 | Train Loss: 0.2967 | Val Loss: 0.2851 | Val Recall: 0.3432 | Val Precision: 0.6598 |Val F1: 0.4515 |Val Accuracy: 0.9342
Validation Loss improved from 0.3252 to 0.2851.
Epoch 6/30 | Train Loss: 0.2590 | Val Loss: 0.2589 | Val Recall: 0.4489 | Val Precision: 0.7113 |Val F1: 0.5504 |Val Accuracy: 0.9424
Validation Loss improved from 0.2851 to 0.2589.
Epoch 7/30 | Train Loss: 0.2344 | Val Loss: 0.2412 | Val Recall: 0.4897 | Val Precision: 0.7173 |Val F1: 0.5820 |Val Accuracy: 0.9452
Validation Loss improved from 0.2589 to 0.2412.
Epoch 8/30 | Train Loss: 0.2156 | Val Loss: 0.2291 | Val Recall: 0.5495 | Val Precision: 0.7427 |Val F1: 0.6316 |Val Accuracy: 0.9476
Validation Loss improved from 0.2412 to 0.2291.
Epoch 9/30 | Train Loss: 0.2015 | Val Loss: 0.2129 | Val Recall: 0.5897 | Val Precision: 0.7524 |Val F1: 0.6612 |Val Accuracy: 0.9509
Validation Loss improved from 0.2291 to 0.2129.
Epoch 10/30 | Train Loss: 0.1908 | Val Loss: 0.2122 | Val Recall: 0.5794 | Val Precision: 0.7961 |Val F1: 0.6707 |Val Accuracy: 0.9507
Validation Loss improved from 0.2129 to 0.2122.
Epoch 11/30 | Train Loss: 0.1843 | Val Loss: 0.2031 | Val Recall: 0.6309 | Val Precision: 0.7837 |Val F1: 0.6990 |Val Accuracy: 0.9539
Validation Loss improved from 0.2122 to 0.2031.
Epoch 12/30 | Train Loss: 0.1753 | Val Loss: 0.2040 | Val Recall: 0.5992 | Val Precision: 0.7971 |Val F1: 0.6841 |Val Accuracy: 0.9520
No improvement in validation loss. Patience: 1
Epoch 13/30 | Train Loss: 0.1700 | Val Loss: 0.1925 | Val Recall: 0.6358 | Val Precision: 0.7817 |Val F1: 0.7012 |Val Accuracy: 0.9537
Validation Loss improved from 0.2031 to 0.1925.
Epoch 14/30 | Train Loss: 0.1633 | Val Loss: 0.1865 | Val Recall: 0.6541 | Val Precision: 0.7867 |Val F1: 0.7143 |Val Accuracy: 0.9553
Validation Loss improved from 0.1925 to 0.1865.
Epoch 15/30 | Train Loss: 0.1584 | Val Loss: 0.1847 | Val Recall: 0.6614 | Val Precision: 0.7741 |Val F1: 0.7133 |Val Accuracy: 0.9557
Validation Loss improved from 0.1865 to 0.1847.
Epoch 16/30 | Train Loss: 0.1545 | Val Loss: 0.1810 | Val Recall: 0.6864 | Val Precision: 0.7926 |Val F1: 0.7357 |Val Accuracy: 0.9571
Validation Loss improved from 0.1847 to 0.1810.
Epoch 17/30 | Train Loss: 0.1490 | Val Loss: 0.1805 | Val Recall: 0.6771 | Val Precision: 0.7944 |Val F1: 0.7311 |Val Accuracy: 0.9565
Validation Loss improved from 0.1810 to 0.1805.
Epoch 18/30 | Train Loss: 0.1458 | Val Loss: 0.1788 | Val Recall: 0.6626 | Val Precision: 0.8049 |Val F1: 0.7268 |Val Accuracy: 0.9562
Validation Loss improved from 0.1805 to 0.1788.
Epoch 19/30 | Train Loss: 0.1426 | Val Loss: 0.1783 | Val Recall: 0.6559 | Val Precision: 0.7826 |Val F1: 0.7137 |Val Accuracy: 0.9555
Validation Loss improved from 0.1788 to 0.1783.
```

```
Epoch 20/30 | Train Loss: 0.1376 | Val Loss: 0.1726 | Val Recall: 0.6951 | Val Precision: 0.7834 |Val F1: 0.7366 |Val Accuracy: 0.9575
Validation Loss improved from 0.1783 to 0.1726.
Epoch 21/30 | Train Loss: 0.1343 | Val Loss: 0.1691 | Val Recall: 0.7048 | Val Precision: 0.7963 |Val F1: 0.7478 |Val Accuracy: 0.9592
Validation Loss improved from 0.1726 to 0.1691.
Epoch 22/30 | Train Loss: 0.1297 | Val Loss: 0.1684 | Val Recall: 0.6832 | Val Precision: 0.8022 |Val F1: 0.7379 |Val Accuracy: 0.9581
Validation Loss improved from 0.1691 to 0.1684.
Epoch 23/30 | Train Loss: 0.1275 | Val Loss: 0.1758 | Val Recall: 0.6458 | Val Precision: 0.8088 |Val F1: 0.7182 |Val Accuracy: 0.9559
No improvement in validation loss. Patience: 1
Epoch 24/30 | Train Loss: 0.1237 | Val Loss: 0.1661 | Val Recall: 0.7106 | Val Precision: 0.8079 |Val F1: 0.7561 |Val Accuracy: 0.9601
Validation Loss improved from 0.1684 to 0.1661.
Epoch 25/30 | Train Loss: 0.1214 | Val Loss: 0.1675 | Val Recall: 0.6864 | Val Precision: 0.8047 |Val F1: 0.7408 |Val Accuracy: 0.9576
No improvement in validation loss. Patience: 1
Epoch 26/30 | Train Loss: 0.1180 | Val Loss: 0.1798 | Val Recall: 0.6531 | Val Precision: 0.8051 |Val F1: 0.7212 |Val Accuracy: 0.9556
No improvement in validation loss. Patience: 2
Epoch 27/30 | Train Loss: 0.1142 | Val Loss: 0.1646 | Val Recall: 0.6978 | Val Precision: 0.8077 |Val F1: 0.7488 |Val Accuracy: 0.9587
Validation Loss improved from 0.1661 to 0.1646.
Epoch 28/30 | Train Loss: 0.1136 | Val Loss: 0.1625 | Val Recall: 0.7073 | Val Precision: 0.8001 |Val F1: 0.7509 |Val Accuracy: 0.9589
Validation Loss improved from 0.1646 to 0.1625.
Epoch 29/30 | Train Loss: 0.1085 | Val Loss: 0.1641 | Val Recall: 0.7250 | Val Precision: 0.7919 |Val F1: 0.7570 |Val Accuracy: 0.9601
```

```
# plot evaluation metrics
plot_training_metrics(history_v8)
```

```
/var/folders/qw/2jrbyvsn4tnckwl2l1rgl1wm0000gp/T/ipykernel_54182/4064934705.py:20: UserWarning: linestyle is redundantly defined by the 'linest
  plt.plot(epochs_range, history['accuracy'], 'k-x', label='Accuracy', linestyle='--')
```



Observation: The validation loss stopped improving for 2 epoches twice but then improved again, so no early stopping was triggered. The epoch 29 gave us the best model with: Train Loss: 0.1085 Val Loss: 0.1641 Val Recall: 72.50% Val Precision: 79.19% Val F1: 75.70% Val Accuracy: 96.01%

Evaluating this best bi-lstm-model on test set.

```
lstm_ner_metrics = evaluate_epoch(lstm_nre_v8, test_loader, label_list)
```

```
print(f"Testing Loss: {lstm_ner_metrics['val_loss']}")
print(f"Accuracy: {(lstm_ner_metrics['accuracy'] * 100):.4f}% - tokens were correctly classified")
print(f"Precision: {(lstm_ner_metrics['precision'] * 100):.4f}% - were correctly redacted out of all redacted tokens")
print(f"Recall: {(lstm_ner_metrics['recall'] * 100):.4f}% - tokens were correctly identified for redaction")
print(f"F1: {(lstm_ner_metrics['f1'] * 100):.4f}% - is the readability score")
```

```
Testing Loss: 0.17113963535853793
Accuracy: 95.6999% - tokens were correctly classified
Precision: 80.1607% - were correctly redacted out of all redacted tokens
Recall: 69.4476% - tokens were correctly identified for redaction
F1: 74.4206% - is the readability score
```

5. LegalBERT Finetuning (Liza)

Using legalBERT tokenizer

```
#add for legalBERT
# tokenizer_legalbert = AutoTokenizer.from_pretrained("distilbert-base-uncased")
# 5. Legal-BERT Finetuning (Liza)

#!pip install seqeval evaluate transformers datasets

import numpy as np
import pandas as pd
import evaluate
from functools import partial
from datasets import Dataset, DatasetDict
from transformers import AutoTokenizer, AutoModelForTokenClassification, TrainingArguments, Trainer, DataCollatorForTokenClassification

# going with legal-bert since it handles domain specific terms better than vanilla bert
model_checkpoint = "nlpaueb/legal-bert-base-uncased"
tokenizer_legal = AutoTokenizer.from_pretrained(model_checkpoint)
metric = evaluate.load("seqeval")

# standard bio tags
label_list = ["O", "B-PER", "I-PER", "B-LOC", "I-LOC", "B-ORG", "I-ORG"]
label2id = {label: i for i, label in enumerate(label_list)}
id2label = {i: label for i, label in enumerate(label_list)}

# just mapping raw types to our simplified schema
type_mapper = {"PERSON": "PER", "ORGANIZATION": "ORG", "LOCATION": "LOC", "ORG": "ORG", "LOC": "LOC", "PER": "PER"}

def tokenize_and_align_labels(examples, tokenizer):
    # the main tricky part: raw data gives us char indices (start=10, end=15),
    # but bert sees tokens. simply splitting by space fails on punctuation (e.g. "Ivanov,").
    # so i'm using offset_mapping from the tokenizer to align labels correctly.


    tokenized_inputs = tokenizer(
        examples["text"], truncation=True, max_length=512,
        return_offsets_mapping=True, padding="max_length"
    )
    labels = []

    for i, doc_offsets in enumerate(tokenized_inputs["offset_mapping"]):
        doc_mentions = examples["entity_mentions"][i] if examples["entity_mentions"][i] is not None else []

        doc_labels = [0] * len(doc_offsets)
        for idx, (start, end) in enumerate(doc_offsets):
            if start == end:
                doc_labels[idx] = -100 # ignore special tokens
                continue

            for mention in doc_mentions:
                if start >= mention['start_offset'] and end <= mention['end_offset']:
                    raw_type = mention['entity_type']
                    short_type = type_mapper.get(raw_type, "ORG")

                    # logic for B- vs I- tags
                    prefix = "B-" if start == mention['start_offset'] else "I-"
                    label_name = f"{prefix}{short_type}"
                    doc_labels[idx] = label2id.get(label_name, 0)
                    break
        labels.append(doc_labels)

    tokenized_inputs["labels"] = labels
    tokenized_inputs.pop("offset_mapping") # we don't need this for training
    return tokenized_inputs

# using the shared dataframes (df_train etc) loaded at the top of the notebook
# converting them to hf dataset format and applying the alignment fix
print("prepping data for legal-bert...")

raw_datasets = DatasetDict({
    "train": Dataset.from_pandas(df_train),
    "validation": Dataset.from_pandas(df_validation),
    "test": Dataset.from_pandas(df_test)
})

tokenized_datasets = raw_datasets.map(
    partial(tokenize_and_align_labels, tokenizer=tokenizer_legal),
    batched=True,
    remove_columns=raw_datasets["train"].column_names
)
print("done. tokens aligned.")
```

```
/home/fred/anaconda3/envs/LDR-NER/lib/python3.14/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipy
  from .autonotebook import tqdm as notebook_tqdm
prepping data for legal-bert...
Map: 100%|██████████| 1112/1112 [00:02<00:00, 396.29 examples/s]
Map: 100%|██████████| 541/541 [00:00<00:00, 605.60 examples/s]
Map: 100%|██████████| 555/555 [00:00<00:00, 709.81 examples/s]done. tokens aligned.
```

```
# initializing the model
model = AutoModelForTokenClassification.from_pretrained(
    model_checkpoint,
    num_labels=len(label_list),
    id2label=id2label,
    label2id=label2id
)

# standard hyperparams for bert fine-tuning.
# 3 epochs is usually enough for transfer learning to converge.
args = TrainingArguments(
    "legal-bert-ner",
    eval_strategy="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,
    save_strategy="no",
    logging_steps=50
)

# using the Trainer api to handle the training loop efficiently
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=DataCollatorForTokenClassification(tokenizer_legal)
)

print("starting training...")
trainer.train()
```

```
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
Loading weights: 100%|██████████| 197/197 [00:00<00:00, 872.75it/s, Materializing param=bert.encoder.layer.11.output.dense.weight]
BertForTokenClassification LOAD REPORT from: nlpaueb/legal-bert-base-uncased
Key                                         | Status     |
--------------------------------------------+------------+-
cls.predictions.transform.dense.weight      | UNEXPECTED |
cls.predictions.transform.LayerNorm.bias    | UNEXPECTED |
cls.seq_relationship.weight                 | UNEXPECTED |
cls.predictions.decoder.bias                | UNEXPECTED |
cls.predictions.transform.LayerNorm.weight  | UNEXPECTED |
cls.predictions.bias                        | UNEXPECTED |
bert.pooler.dense.weight                    | UNEXPECTED |
bert.pooler.dense.bias                      | UNEXPECTED |
cls.seq_relationship.bias                   | UNEXPECTED |
cls.predictions.transform.dense.bias        | UNEXPECTED |
cls.predictions.decoder.weight              | UNEXPECTED |
classifier.weight                           | MISSING    |
classifier.bias                             | MISSING    |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
- MISSING   :those params were newly initialized because missing from the checkpoint. Consider training on your downstream task.
starting training...
```

[210/210 29:31, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.504909	0.133475
2	0.134588	0.104182
3	0.091035	0.101755

```
TrainOutput(global_step=210, training_loss=0.2026848520551409, metrics={'train_runtime': 1773.6059, 'train_samples_per_second': 1.881,
```

```
def evaluate_model_performance(trainer, eval_dataset, model_name="MyModel"):
    # getting predictions and filtering out the -100 ignored tokens
    # to calculate real metrics
    print(f"--- evaluating {model_name} ---")
    predictions, labels, _ = trainer.predict(eval_dataset)
    predictions = np.argmax(predictions, axis=2)

    true_predictions = [
```

```
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    results = metric.compute(predictions=true_predictions, references=true_labels)

    return {
        "Model": model_name,
        "F1 Score": results['overall_f1'],
        # recall is critical here - we can't miss sensitive info
        "Recall": results['overall_recall'],
        "Precision": results['overall_precision'],
        "Accuracy": results['overall_accuracy']
    }

metrics_legal = evaluate_model_performance(trainer, tokenized_datasets["test"], model_name="Legal-BERT (Liza)")

print("\n final results:")
display(pd.DataFrame([metrics_legal]).round(4))
```

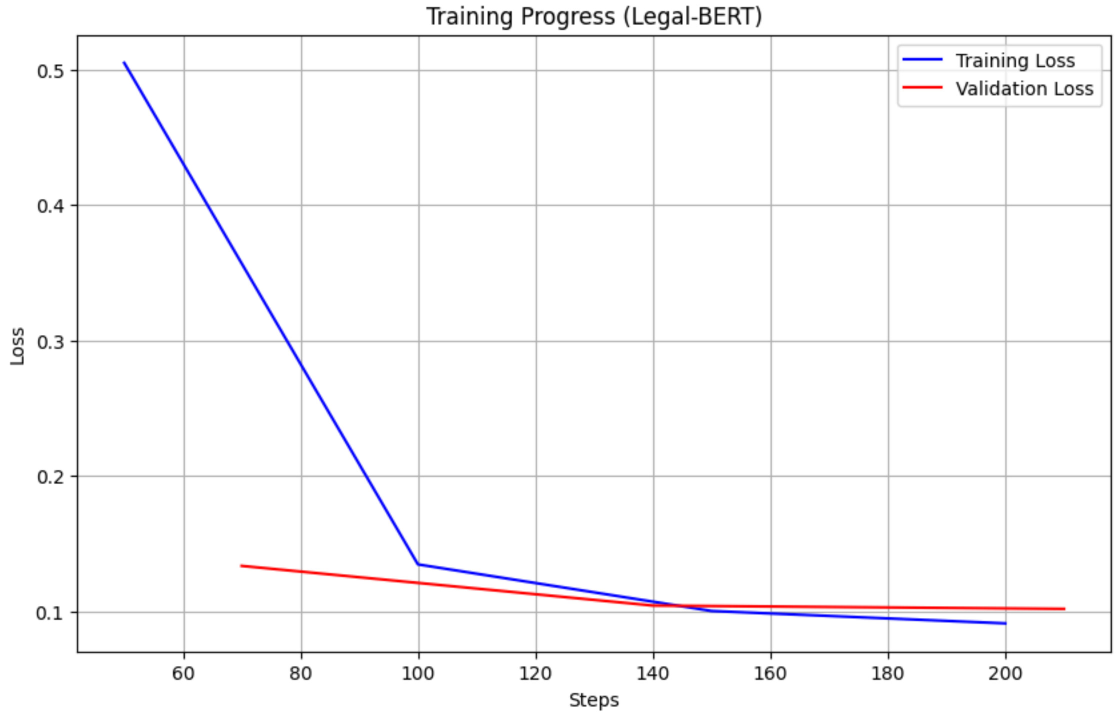
```
--- evaluating Legal-BERT (Liza) ---

 final results:
```

	Model	F1 Score	Recall	Precision	Accuracy
0	Legal-BERT (Liza)	0.8192	0.8393	0.8001	0.9651

```
# train_legalbert = preprocess_data(df_train ,tokenizer_legalbert)
# test_legalbert = preprocess_data(df_test, tokenizer_legalbert)
# validation_legalbert = preprocess_data(df_validation, tokenizer_legalbert)
```

```
import matplotlib.pyplot as plt
import json
import os

output_dir = "./legal_bert_output"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

print(f"Saving model to {output_dir}...")
trainer.save_model(output_dir)
tokenizer_legal.save_pretrained(output_dir)

history = trainer.state.log_history
history_path = f"{output_dir}/training_history.json"
with open(history_path, "w") as f:
    json.dump(history, f)
print(f"History saved to {history_path}")

train_loss = [x['loss'] for x in history if 'loss' in x]
steps = [x['step'] for x in history if 'loss' in x]

if train_loss:
    plt.figure(figsize=(10, 6))
    plt.plot(steps, train_loss, label="Training Loss", color="blue")

    val_loss = [x['eval_loss'] for x in history if 'eval_loss' in x]
    val_steps = [x['step'] for x in history if 'eval_loss' in x]
    if val_loss:
        plt.plot(val_steps, val_loss, label="Validation Loss", color="red")

    plt.xlabel("Steps")
    plt.ylabel("Loss")
    plt.title("Training Progress (Legal-BERT)")
    plt.legend()
    plt.grid(True)

    graph_path = f"{output_dir}/training_graph.png"
    plt.savefig(graph_path)
    print(f"Graph saved to {graph_path}")
    plt.show()
else:
    print("Not enough history to plot graph yet.")
```

```
Saving model to ./legal_bert_output...
Writing model shards: 100%|██████████| 1/1 [00:01<00:00,  1.04s/it]
History saved to ./legal_bert_output/training_history.json
Graph saved to ./legal_bert_output/training_graph.png
```



6. DistilBERT (Ritvik & Mit)

Using tokenized datasets: train_distilbert, test_distilbert, validation_distilbert

```
import numpy as np
import pandas as pd
import torch
from datasets import Dataset, DatasetDict
from transformers import AutoTokenizer, AutoModelForTokenClassification, TrainingArguments, Trainer, DataCollatorForTokenClassification
from transformers import DistilBertForTokenClassification

"""
setting up distilbert as our baseline model.
it is lighter and faster than legal-bert and has fewer parameters.

"""
distil_checkpoint = "distilbert-base-uncased" # letter case independent
distil_tokenizer = AutoTokenizer.from_pretrained(distil_checkpoint) # using autotokenizer

label_list = ["O", "B-PER", "I-PER", "B-LOC", "I-LOC", "B-ORG", "I-ORG"]
label2id = {label: i for i, label in enumerate(label_list)}
id2label = {i: label for i, label in enumerate(label_list)}
type_mapper = {"PERSON": "PER", "ORGANIZATION": "ORG", "LOCATION": "LOC", "ORG": "ORG", "LOC": "LOC", "PER": "PER"}
```

```
import torch
print(f"Is GPU available? {torch.cuda.is_available()}")
print(f"Device Name: {torch.cuda.get_device_name(0)}")
```

```
Is GPU available? True
Device Name: NVIDIA GeForce RTX 3050 Laptop GPU
```

```
def tokenize_and_align_distil(examples):

    """
        Instead of relying on spaces and commas to create tokens, we are using offset mapping
        To make sure that the tokens align well with the BERT model and they come from the correct offsets in the dataset

    """

    tokenized_inputs = distil_tokenizer(
        examples["text"], truncation=True, max_length=512,
        return_offsets_mapping=True, padding="max_length" # padding the tokens with max token length
    )
    labels = []
```

```
    for i, doc_offsets in enumerate(tokenized_inputs["offset_mapping"]):
        doc_mentions = examples["entity_mentions"][i] if examples["entity_mentions"][i] is not None else []

        doc_labels = [0] * len(doc_offsets)
        for idx, (start, end) in enumerate(doc_offsets):
            if start == end:
                doc_labels[idx] = -100 # ignore special tokens
                continue

            for mention in doc_mentions:
                if start >= mention['start_offset'] and end <= mention['end_offset']:
                    raw_type = mention['entity_type']
                    short_type = type_mapper.get(raw_type, "ORG")

                    # logic to check beginning and inside of tokens
                    prefix = "B-" if start == mention['start_offset'] else "I-"
                    label_name = f"{prefix}{short_type}"
                    doc_labels[idx] = label2id.get(label_name, 0)
                    break
        labels.append(doc_labels)

    tokenized_inputs["labels"] = labels
    tokenized_inputs.pop("offset_mapping")
    return tokenized_inputs


# creating a dataset dict
distil_datasets = DatasetDict({
    "train": Dataset.from_pandas(df_train),
    "validation": Dataset.from_pandas(df_validation),
    "test": Dataset.from_pandas(df_test)
})

distil_tokenized = distil_datasets.map(
    tokenize_and_align_distil,
    batched=True,
    remove_columns=distil_datasets["train"].column_names
)
print("done. distilbert tokens aligned.")
```

```
Map: 100%|██████████| 1112/1112 [00:05<00:00, 213.66 examples/s]
Map: 100%|██████████| 541/541 [00:01<00:00, 341.72 examples/s]
Map: 100%|██████████| 555/555 [00:01<00:00, 343.58 examples/s]done. distilbert tokens aligned.
```

Baseline DistilBERT Model (kesha)

```
# initializing the distilbert model
distil_model = AutoModelForTokenClassification.from_pretrained(
    distil_checkpoint,
    num_labels=len(label_list),
    id2label=id2label,
    label2id=label2id
)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
distil_model.to(device)

# setting hyperparams
db_args = TrainingArguments(
    "distilbert-ner-output",
    eval_strategy="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,
    save_strategy="no",
    logging_steps=50,
    seed=42,
    data_seed=42,
)

db_trainer = Trainer(
    model=distil_model,
    args=db_args,
    train_dataset=distil_tokenized["train"],
    eval_dataset=distil_tokenized["validation"],
    data_collator=DataCollatorForTokenClassification(distil_tokenizer)
)
```

```
print("starting distilbert training...")
db_trainer.train()
```

```
Loading weights: 100%|██████████| 100/100 [00:00<00:00, 543.76it/s, Materializing param=distilbert.transformer.layer.5.sa_layer_norm.weight]
DistilBertForTokenClassification LOAD REPORT from: distilbert-base-uncased
Key                     | Status     |
------------------------+------------+-
vocab_transform.weight  | UNEXPECTED |
vocab_layer_norm.weight | UNEXPECTED |
vocab_transform.bias    | UNEXPECTED |
vocab_layer_norm.bias   | UNEXPECTED |
vocab_projector.bias    | UNEXPECTED |
classifier.weight       | MISSING    |
classifier.bias         | MISSING    |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
- MISSING   :those params were newly initialized because missing from the checkpoint. Consider training on your downstream task.
starting distilbert training...
```

[210/210 03:12, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.525557	0.161509
2	0.159323	0.118078
3	0.103047	0.118716

```
TrainOutput(global_step=210, training_loss=0.22058556817826772, metrics={'train_runtime': 194.0184, 'train_samples_per_second': 17.194,
```

Model Evaluation

```
import evaluate
import numpy as np
import pandas as pd

# initialize metric
metric = evaluate.load("seqeval")

def evaluate_model_performance(trainer, eval_dataset, model_name):
    # getting predictions and filtering out the -100 ignored tokens
    # to calculate real metrics
    print(f"--- evaluating {model_name} ---")
    predictions, labels, _ = trainer.predict(eval_dataset)
    predictions = np.argmax(predictions, axis=2)

    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    results = metric.compute(predictions=true_predictions, references=true_labels)

    return {
        "Model": model_name,
        "F1 Score": results['overall_f1'],
        # recall is critical here as we can't miss sensitive info
        "Recall": results['overall_recall'],
        "Precision": results['overall_precision'],
        "Accuracy": results['overall_accuracy']
    }

metrics_db = evaluate_model_performance(db_trainer, distil_tokenized["test"], model_name="DistilBERT_v1")
```

```
--- evaluating DistilBERT_v1 ---
```

```
# evaluation Plots
print("\n final results (DistilBERT):")
display(pd.DataFrame([metrics_db]).round(4))

# saving and plotting
import matplotlib.pyplot as plt
import json
import os

output_dir = "./distilbert_output_v2"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
```

```
print(f"saving model to {output_dir}...")
db_trainer.save_model(output_dir)
distil_tokenizer.save_pretrained(output_dir)

history = db_trainer.state.log_history
# filtering for loss values to plot
train_loss = [x['loss'] for x in history if 'loss' in x]
steps = [x['step'] for x in history if 'loss' in x]

if train_loss:
    plt.figure(figsize=(10, 6))
    plt.plot(steps, train_loss, label="Training Loss (DistilBERT)", color="Red")

    # checking for validation loss
    val_loss = [x['eval_loss'] for x in history if 'eval_loss' in x]
    val_steps = [x['step'] for x in history if 'eval_loss' in x]

    if val_loss:
        plt.plot(val_steps, val_loss, label="Validation Loss", color="Green")

    plt.xlabel("Steps")
    plt.ylabel("Loss")
    plt.title("Training Progress (DistilBERT)")
    plt.legend()
    plt.grid(True)
    plt.show()
```

final results (DistilBERT):

	Model	F1 Score	Recall	Precision	Accuracy
0	DistilBERT_v1	0.7982	0.8173	0.78	0.9587

```
saving model to ./distilbert_output_v2...
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  2.61it/s]
```

Observation: The model accuracy is quite high (0.95), while the recall is low (0.82). Since our dataset has mostly 'O' in places of non-sensitive words, the model becomes lazy and predicts sensitive words as non-sensitive to gain a higher accuracy. This is also critical because we are working on legal data and in this domain, it is very costly for the model to miss sensitive words.

Fine-Tuning strategy 1: Implement Early Stopping and Label Smoothing

DistilBERT Fine-Tuned v2 (Ritvik)

```
from transformers import EarlyStoppingCallback
# initializing the distilbert model
distil_model_v2 = AutoModelForTokenClassification.from_pretrained(
    distil_checkpoint,
    num_labels=len(label_list),
    id2label=id2label,
    label2id=label2id
)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
distil_model_v2.to(device)

# setting hyperparams
db_args = TrainingArguments(
    "distilbert-ner-v2-output",
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=2e-5,
    num_train_epochs=5, # increased Epochs to train more
    weight_decay=0.05, # increased to reduce memorisation
    logging_steps=50,
    metric_for_best_model="loss",
    label_smoothing_factor=0.1, # helps with class imbalance problem as it gives less weights to each label
    seed=42,
    data_seed=42
)

db_trainer_v2 = Trainer(
    model=distil_model_v2,
    args=db_args,
    train_dataset=distil_tokenized["train"],
    eval_dataset=distil_tokenized["validation"],
    data_collator=DataCollatorForTokenClassification(distil_tokenizer),
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]
)

print("starting distilbert v2 training...")
db_trainer_v2.train()
metrics_db = evaluate_model_performance(db_trainer_v2, distil_tokenized["test"], model_name="DistilBERT_v2")
print("\n final results (DistilBERT):")
display(pd.DataFrame([metrics_db]).round(4))
```

```
Loading weights: 100%|██████████| 100/100 [00:00<00:00, 517.87it/s, Materializing param=distilbert.transformer.layer.5.sa_layer_norm.weight]
DistilBertForTokenClassification LOAD REPORT from: distilbert-base-uncased
Key                     | Status     |
------------------------+------------+-
vocab_transform.weight  | UNEXPECTED |
vocab_layer_norm.weight | UNEXPECTED |
vocab_transform.bias    | UNEXPECTED |
vocab_layer_norm.bias   | UNEXPECTED |
vocab_projector.bias    | UNEXPECTED |
classifier.weight       | MISSING    |
classifier.bias         | MISSING    |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
- MISSING   :those params were newly initialized because missing from the checkpoint. Consider training on your downstream task.
starting distilbert v2 training...
```

[350/350 38:07, Epoch 5/5]

Epoch	Training Loss	Validation Loss
1	0.839470	0.555659
2	0.550116	0.526227
3	0.514136	0.522698
4	0.514076	0.523622
5	0.503792	0.523969

```
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  4.15it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  3.53it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  3.87it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  3.83it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  4.13it/s]
There were missing keys in the checkpoint model loaded: ['distilbert.embeddings.LayerNorm.weight', 'distilbert.embeddings.LayerNorm.bias'].
There were unexpected keys in the checkpoint model loaded: ['distilbert.embeddings.LayerNorm.beta', 'distilbert.embeddings.LayerNorm.gamma'].
--- evaluating DistilBERT_v2 ---

 final results (DistilBERT):
```

	Model	F1 Score	Recall	Precision	Accuracy
0	DistilBERT_v2	0.806	0.8295	0.7839	0.9605

```
# saving and plotting
import matplotlib.pyplot as plt
import json
import os

output_dir = "./distilbert_output_fine-tuned_v2"
```

```
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    print(f"saving model to {output_dir}...")
    db_trainer_v2.save_model(output_dir)
    distil_tokenizer.save_pretrained(output_dir)

    history = db_trainer_v2.state.log_history
    # filtering for loss values to plot
    train_loss = [x['loss'] for x in history if 'loss' in x]
    steps = [x['step'] for x in history if 'loss' in x]

    if train_loss:
        plt.figure(figsize=(10, 6))
        plt.plot(steps, train_loss, label="Training Loss (DistilBERT)", color="Red")

        # checking for validation loss
        val_loss = [x['eval_loss'] for x in history if 'eval_loss' in x]
        val_steps = [x['step'] for x in history if 'eval_loss' in x]

        if val_loss:
            plt.plot(val_steps, val_loss, label="Validation Loss", color="Green")

        plt.xlabel("Steps")
        plt.ylabel("Loss")
        plt.title("Training Progress (DistilBERT v2)")
        plt.legend()
        plt.grid(True)
        plt.show()
```

```
saving model to ./distilbert_output_fine-tuned_v2...
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  2.58it/s]
```

Observations: Label Smoothing had a positive impact on the model's recall, which increased from 81.7 to 82.9. The F1 score also has a slight increase, however the precision dropped. This is because the model is now not as aggressive into predicting a class with high confidence, instead we tell the model to attempt to predict with lower confidence, which makes it more cautious. This is imporant because for Legal documentation, it is critical to not miss any sensitive information.

DistilBERT Fine-Tuned v3 (Mit)

```
from transformers import EarlyStoppingCallback
# initializing the distilbert model
distil_model_v3 = AutoModelForTokenClassification.from_pretrained(
    distil_checkpoint,
    num_labels=len(label_list),
    id2label=id2label,
    label2id=label2id
)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
distil_model_v3.to(device)
```

```python
# setting hyperparams
db_args_v3 = TrainingArguments(
    "distilbert-ner-v3-output",
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=3e-5, # increased learning rate
    num_train_epochs=5, # increased Epochs to train more
    weight_decay=0.01,  # reduced weight decay to make the model more aggressive, the model does not need very high weights to make a predict:
    logging_steps=50,
    metric_for_best_model="loss",
    seed=42,
    data_seed=42
)

db_trainer_v3 = Trainer(
    model=distil_model_v3,
    args=db_args_v3,
    train_dataset=distil_tokenized["train"],
    eval_dataset=distil_tokenized["validation"],
    data_collator=DataCollatorForTokenClassification(distil_tokenizer),
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]
)

print("starting distilbert v3 training...")
db_trainer_v3.train()
metrics_db_v3 = evaluate_model_performance(db_trainer_v3, distil_tokenized["test"], model_name="DistilBERT_v3")
print("\n final results (DistilBERT):")
display(pd.DataFrame([metrics_db]).round(4))
```

```
Loading weights: 100%|██████████| 100/100 [00:00<00:00, 677.74it/s, Materializing param=distilbert.transformer.layer.5.sa_layer_norm.weight]
DistilBertForTokenClassification LOAD REPORT from: distilbert-base-uncased
Key                     | Status     |
------------------------+------------+-
vocab_transform.weight  | UNEXPECTED |
vocab_layer_norm.weight | UNEXPECTED |
vocab_transform.bias    | UNEXPECTED |
vocab_layer_norm.bias   | UNEXPECTED |
vocab_projector.bias    | UNEXPECTED |
classifier.weight       | MISSING    |
classifier.bias         | MISSING    |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
- MISSING   :those params were newly initialized because missing from the checkpoint. Consider training on your downstream task.
starting distilbert v3 training...
```

[350/350 54:14, Epoch 5/5]

Epoch	Training Loss	Validation Loss
1	0.497990	0.138706
2	0.128608	0.101452
3	0.086107	0.098404
4	0.084529	0.105449
5	0.069774	0.104834

```
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  3.29it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  3.77it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  3.88it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  3.86it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  3.70it/s]
There were missing keys in the checkpoint model loaded: ['distilbert.embeddings.LayerNorm.weight', 'distilbert.embeddings.LayerNorm.bias'].
There were unexpected keys in the checkpoint model loaded: ['distilbert.embeddings.LayerNorm.beta', 'distilbert.embeddings.LayerNorm.gamma'].
--- evaluating DistilBERT_v3 ---

 final results (DistilBERT):
```

	Model	F1 Score	Recall	Precision	Accuracy
0	DistilBERT_v2	0.806	0.8295	0.7839	0.9605

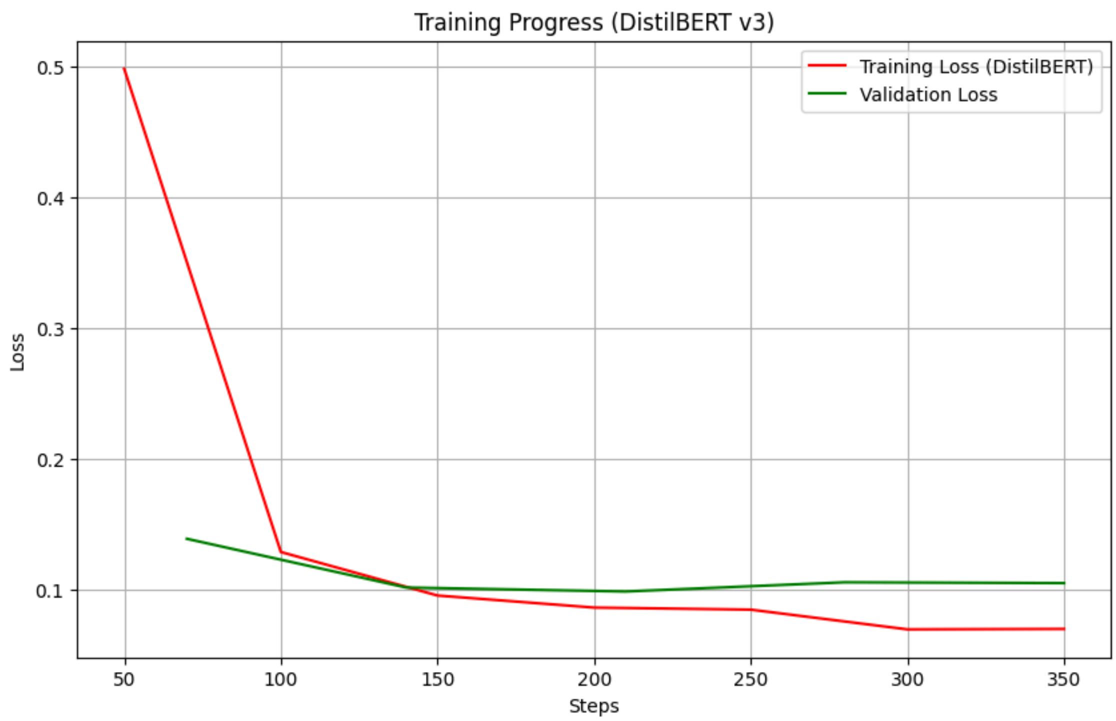
```python
print("\n final results (DistilBERT):")
display(pd.DataFrame([metrics_db_v3]).round(4))
```

```
 final results (DistilBERT):
```

	Model	F1 Score	Recall	Precision	Accuracy
0	DistilBERT_v3	0.8071	0.8384	0.778	0.9626

```python
# saving and plotting
import matplotlib.pyplot as plt
import json
import os

output_dir = "./distilbert_output_fine-tuned_v3"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

print(f"saving model to {output_dir}...")
db_trainer_v3.save_model(output_dir)
distil_tokenizer.save_pretrained(output_dir)

history = db_trainer_v3.state.log_history
# filtering for loss values to plot
train_loss = [x['loss'] for x in history if 'loss' in x]
steps = [x['step'] for x in history if 'loss' in x]

if train_loss:
    plt.figure(figsize=(10, 6))
    plt.plot(steps, train_loss, label="Training Loss (DistilBERT)", color="Red")

    # checking for validation loss
    val_loss = [x['eval_loss'] for x in history if 'eval_loss' in x]
    val_steps = [x['step'] for x in history if 'eval_loss' in x]

    if val_loss:
        plt.plot(val_steps, val_loss, label="Validation Loss", color="Green")

    plt.xlabel("Steps")
    plt.ylabel("Loss")
    plt.title("Training Progress (DistilBERT v3)")
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
saving model to ./distilbert_output_fine-tuned_v3...
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  2.73it/s]
```



Observations:

8. ALBERT finetuning (Sayeed)

```python
# ALBERT Finetuning (Sayeed)
```

```python
# using the shared dataframes (df_train etc) loaded at the top of the notebook
# converting them to hf dataset format and applying the alignment fix
print("prepping data for ALBERT...")

tokenizer_albert = AutoTokenizer.from_pretrained("albert-base-v2", use_fast=True)
tokenized_datasets = raw_datasets.map(
```

```
    partial(tokenize_and_align_labels, tokenizer=tokenizer_albert),
    batched=True,
    remove_columns=raw_datasets["train"].column_names
)
print("done. tokens aligned.")
```

```
prepping data for ALBERT...
Map: 100%|██████████| 1112/1112 [00:02<00:00, 406.38 examples/s]
Map: 100%|██████████| 541/541 [00:00<00:00, 595.72 examples/s]
Map: 100%|██████████| 555/555 [00:00<00:00, 579.72 examples/s]done. tokens aligned.
```

```
import torch

def sanity_forward_pass(model, dataset, n=2):
    model = model.to("cpu")
    model.train()

    batch = {
        "input_ids": torch.tensor([dataset[i]["input_ids"] for i in range(n)], device="cpu"),
        "attention_mask": torch.tensor([dataset[i]["attention_mask"] for i in range(n)], device="cpu"),
        "labels": torch.tensor([dataset[i]["labels"] for i in range(n)], device="cpu"),
    }

    out = model(**batch)
    print("Forward OK. Loss:", float(out.loss))

sanity_forward_pass(model, tokenized_datasets["train"], n=2)
```

```
Forward OK. Loss: 2.0854053497314453
/tmp/ipykernel_9036/1652176267.py:14: UserWarning: Converting a tensor with requires_grad=True to a scalar may lead to unexpected behavior.
Consider using tensor.detach() first. (Triggered internally at /pytorch/torch/csrc/autograd/generated/python_variable_methods.cpp:836.)
  print("Forward OK. Loss:", float(out.loss))
```

```
import os
from transformers import (
    AutoModelForTokenClassification,
    TrainingArguments,
    Trainer,
    DataCollatorForTokenClassification,
    EarlyStoppingCallback
)

# Initialize model
model = AutoModelForTokenClassification.from_pretrained(
    "albert-base-v2",
    num_labels=len(label_list),
    id2label=id2label,
    label2id=label2id
)

# Training args (Early stopping requires eval+save each epoch + load best)
args = TrainingArguments(
    output_dir="albert-ner",
    eval_strategy="epoch",
    save_strategy="epoch",             # must save checkpoints for early stopping / best model
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
    greater_is_better=False,

    learning_rate=2e-5,
    warmup_ratio=0.1,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=10,               # allow early stopping to stop early
    weight_decay=0.01,

    logging_strategy="steps",
    logging_steps=50,

    save_total_limit=2,
    report_to="none"
)

# Trainer
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=DataCollatorForTokenClassification(tokenizer_albert),
```

```
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]
)

print("starting training...")
trainer.train()

print("Best checkpoint:", trainer.state.best_model_checkpoint)


best_dir = os.path.join(args.output_dir, "best_model")
trainer.save_model(best_dir)
tokenizer_albert.save_pretrained(best_dir)
print("Saved best model to:", best_dir)


metrics_albert = evaluate_model_performance(
    trainer,
    tokenized_datasets["test"],
    model_name="ALBERT (Fine-tuned)"
)

print("\nfinal results:")
display(pd.DataFrame([metrics_albert]).round(4))
```

```
Loading weights: 100%|██████████| 23/23 [00:00<00:00, 630.37it/s, Materializing param=albert.encoder.embedding_hidden_mapping_in.weight]
AlbertForTokenClassification LOAD REPORT from: albert-base-v2
Key                          | Status     |
-----------------------------+------------+-
predictions.LayerNorm.bias   | UNEXPECTED |
predictions.dense.weight     | UNEXPECTED |
predictions.decoder.bias     | UNEXPECTED |
predictions.dense.bias       | UNEXPECTED |
albert.pooler.bias           | UNEXPECTED |
predictions.bias             | UNEXPECTED |
albert.pooler.weight         | UNEXPECTED |
predictions.LayerNorm.weight | UNEXPECTED |
classifier.bias              | MISSING    |
classifier.weight            | MISSING    |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
- MISSING   :those params were newly initialized because missing from the checkpoint. Consider training on your downstream task.
warmup_ratio is deprecated and will be removed in v5.2. Use `warmup_steps` instead.
starting training...
```

[350/700 1:36:15 < 1:36:49, 0.06 it/s, Epoch 5/10]

Epoch	Training Loss	Validation Loss
1	0.950470	0.134053
2	0.125220	0.092737
3	0.080617	0.091702
4	0.080936	0.097672
5	0.062784	0.095690

```
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  7.08it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  9.01it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  7.16it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00, 11.82it/s]
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  8.36it/s]
There were missing keys in the checkpoint model loaded: ['albert.embeddings.LayerNorm.weight', 'albert.embeddings.LayerNorm.bias', 'albert.en
There were unexpected keys in the checkpoint model loaded: ['albert.embeddings.LayerNorm.beta', 'albert.embeddings.LayerNorm.gamma', 'albert.
Best checkpoint: albert-ner/checkpoint-210
Writing model shards: 100%|██████████| 1/1 [00:00<00:00,  3.28it/s]
Saved best model to: albert-ner/best_model
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 63
     59 tokenizer_albert.save_pretrained(best_dir)
     60 print("Saved best model to:", best_dir)
---> 63 metrics_albert = evaluate_model_performance(
     64     trainer,
     65     tokenized_datasets["test"],
     66     model_name="ALBERT (Fine-tuned)"
     67 )
     69 print("\nfinal results:")
     70 display(pd.DataFrame([metrics_albert]).round(4))

NameError: name 'evaluate_model_performance' is not defined
```

```
import numpy as np

def evaluate_model_performance(trainer, eval_dataset, label_list, metric, model_name="MyModel"):
    print(f"--- evaluating {model_name} ---")
```

```
    predictions, labels, _ = trainer.predict(eval_dataset)
    predictions = np.argmax(predictions, axis=2)

    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    results = metric.compute(predictions=true_predictions, references=true_labels)

    return {
        "Model": model_name,
        "F1 Score": results["overall_f1"],
        "Recall": results["overall_recall"],
        "Precision": results["overall_precision"],
        "Accuracy": results["overall_accuracy"],
    }
```

```
metrics_albert = evaluate_model_performance(
    trainer=trainer,
    eval_dataset=tokenized_datasets["test"],
    label_list=label_list,
    metric=metric,
    model_name="ALBERT (Fine-tuned)"
)

print("\nfinal results:")
display(pd.DataFrame([metrics_albert]).round(4))
```

```
--- evaluating ALBERT (Fine-tuned) ---

final results:
```

	Model	F1 Score	Recall	Precision	Accuracy
0	ALBERT (Fine-tuned)	0.8161	0.8481	0.7865	0.9643

```
import os, json
import matplotlib.pyplot as plt

def export_best_and_plot(trainer, tokenizer, output_dir="albert-ner", export_dirname="best_model", title="Training Progress"):
    # 1) Export best model to a clean folder
    best_export_dir = os.path.join(output_dir, export_dirname)
    os.makedirs(best_export_dir, exist_ok=True)

    # Trainer usually loads best model at end if load_best_model_at_end=True
    trainer.save_model(best_export_dir)
    tokenizer.save_pretrained(best_export_dir)
    print("Exported best model to:", best_export_dir)

    # 2) Save training history
    history = trainer.state.log_history
    history_path = os.path.join(output_dir, "training_history.json")
    with open(history_path, "w", encoding="utf-8") as f:
        json.dump(history, f, indent=2)
    print("Saved history to:", history_path)

    # 3) Plot train/val loss (epoch if available else step)
    use_epoch = any("epoch" in x for x in history)
    x_key = "epoch" if use_epoch else "step"

    train_x = [x[x_key] for x in history if "loss" in x and x_key in x]
    train_loss = [x["loss"] for x in history if "loss" in x and x_key in x]

    val_x = [x[x_key] for x in history if "eval_loss" in x and x_key in x]
    val_loss = [x["eval_loss"] for x in history if "eval_loss" in x and x_key in x]
```