Neural Network For NER

1. Data Management

1.1 Experimental Target

Design a data preprocessing and loading solution for the NER task.

1.2 Data processing

1.2.1 Random seed

- The random seed is fixed to 42 to ensure the repeatability of the experiment.

```
torch.manual_seed(42)
np.random.seed(42)
```

1.2.2 Defining global parameters

- Specify file path, batch size, and maximum length.

```
Data = "assignment3-ner_dataset.txt"
BatchSize = 32
Maxlen = 150
```

1.2.3 Data loading

- Read the file line by line, parse the word-tag pairs, and build a list of sentences.
- Convert a text file to a list of (Word, Label) tuples.
- Implemented original parsing of NER datasets and added exception handling to ensure robustness.

```
First sentence: [('010', 'I-MISC'), ('is', 'O'), ('the', 'O'), ('tenth', 'O')
```

...

```
('Mad', 'I-ORG'), ('Capsule', 'I-ORG'), ('Markets', 'I-ORG'), ('.', 'O')]
```

1.2.4 Building vocabulary and label mapping functions

- Count words and tags, build ID mapping.
- Create vocabulary (WordId) and label mapping (LabelId, IdLabel) to convert text into numeric IDs suitable for model input.

```
WordId: 8506
LabelId: 6
LabelId reflect: {'<PAD>': 0, 'I-LOC': 1, 'I-MISC': 2, 'I-ORG': 3, 'I-PER': 4, 'O': 5}
IdLabel reflect: {0: '<PAD>', 1: 'I-LOC', 2: 'I-MISC', 3: 'I-ORG', 4: 'I-PER', 5: 'O'}
```

1.2.5 Define Dataset

- Convert sentences to sequences of word IDs and tag IDs, truncated to Maxlen. (Unseen words are mapped to <UNK>)
- Returns the word ID tensor, tag ID tensor, and sentence length based on the index.
- At the same time, use the Filling function to fill each sentence into the tensor, and the unfilled part is 0.
- Create a Mask (PadWords != 0) to mark the non-filled position.
- In summary, the above steps convert the input sentence, vocabulary, and label mapping into an ID sequence. By dynamically filling and generating masks, efficient and standardized data input is provided for NER tasks.

1.3 Split the data

```
Number of sentences: 1696
  (010, I-MISC)
  (is, O)
  (the, O)
  (tenth, O)
  (album, O)
  (from, O)
  (Japanese, I-MISC)
  (Punk, O)
  (Techno, O)
  (band, O)
  (The, I-ORG)
  (Mad, I-ORG)
  (Capsule, I-ORG)
  (Markets, I-ORG)
  (., O)
WordId: 8506
LabelId: 6
LabelId reflect: {'<PAD>': 0, 'I-LOC': 1, 'I-MISC': 2, 'I-ORG': 3, 'I-PER': 4, 'O': 5}
IdLabel reflect: {0: '<PAD>', 1: 'I-LOC', 2: 'I-MISC', 3: 'I-ORG', 4: 'I-PER', 5: 'O'}
Sentence length distribution:
Length 1-20: 858 sentences
Length 21-40: 679 sentences
Length 41-60: 131 sentences
Length 61-80: 23 sentences
...
Length 101-120: 0 sentences
Length 121-150: 1 sentences
Length 151-145: 0 sentences
Label distribution: Counter({'O': 32576, 'I-ORG': 1958, 'I-PER': 1634, 'I-LOC': 1447, 'I-MISC': 1392})
```

Based on the results of the preliminary data analysis, we adopted the following experimental method: first, all data were split in a 1:4 ratio, that is, 20% test set and 80% cross-validation set, and the cross-validation set was subjected to 5-fold cross-validation. Reasons:

- The dataset contains 1,696 sentences. Due to the imbalanced label distribution, stratified splitting is used to ensure that the label distribution of the training, validation, and test subsets is consistent, thereby reducing evaluation bias.
- To enhance training diversity and further alleviate overfitting, a 5-fold cross-validation was performed on the 1,357 sentences of training + validation data, with

each sentence being trained 4 times and validated once.

- In summary, this experimental scheme combines stratified splitting and cross-validation to achieve robust evaluation, reliable testing, and efficient data utilization, which is suitable for the requirements of the NER task.

1.3.1 Split steps

- Since the label 'o' accounts for the majority, 'o' is removed when splitting the data and other labels are selected as the main labels of the sentence.

```python
def MainLabel(Sentence, LabelId):
    if not Sentence:
        return LabelId['O']
    Labels = [Label for _,Label in Sentence if Label!='O']
    return LabelId[Labels[0]] if Labels else LabelId['O']
```

- After counting the main tags, the sentence index is divided into a test set (20%) and a training + validation set (80%), using stratification to ensure that the main tags are evenly distributed.
- Finally, the training + validation set is divided into 5 folds and the results are returned

1.3.2 Split Result

```
Split Statistics:
Test Set: 340
Test set label distribution: Counter({'O': 6573, 'I-ORG': 400, 'I-PER': 314, 'I-LOC': 306, 'I-MISC': 293})
Fold1:
  Training set: 1084
  Validation set: 272
  Training set label distribution: Counter({'O': 20788, 'I-ORG': 1268, 'I-PER': 1057, 'I-LOC': 904, 'I-MISC': 898})
  Validation set label distribution: Counter({'O': 5215, 'I-ORG': 290, 'I-PER': 263, 'I-LOC': 237, 'I-MISC': 201})
Fold2:
  Training set: 1085
  Validation set: 271
  Training set label distribution: Counter({'O': 21049, 'I-ORG': 1229, 'I-PER': 1049, 'I-LOC': 892, 'I-MISC': 855})
  Validation set label distribution: Counter({'O': 4954, 'I-ORG': 329, 'I-PER': 271, 'I-LOC': 249, 'I-MISC': 244})
Fold3:
  Training set: 1085
  Validation set: 271
  Training set label distribution: Counter({'O': 20887, 'I-ORG': 1256, 'I-PER': 1069, 'I-LOC': 917, 'I-MISC': 873})
  Validation set label distribution: Counter({'O': 5116, 'I-ORG': 302, 'I-PER': 251, 'I-MISC': 226, 'I-LOC': 224})
Fold4:
  Training set: 1085
  Validation set: 271
  Training set label distribution: Counter({'O': 20565, 'I-ORG': 1194, 'I-PER': 1069, 'I-LOC': 958, 'I-MISC': 865})
  Validation set label distribution: Counter({'O': 5438, 'I-ORG': 364, 'I-PER': 251, 'I-MISC': 234, 'I-LOC': 183})
Fold5:
  Training set: 1085
  Validation set: 271
  Training set label distribution: Counter({'O': 20723, 'I-ORG': 1285, 'I-PER': 1036, 'I-MISC': 905, 'I-LOC': 893})
  Validation set label distribution: Counter({'O': 5280, 'I-PER': 284, 'I-ORG': 273, 'I-LOC': 248, 'I-MISC': 194})
```

```
Test TrainDataLoader (First Fold):
Batch 1:
PadWords shape: torch.Size([32, 80])
PadLabels shape: torch.Size([32, 80])
Mask shape: torch.Size([32, 80])
Lengths shape: torch.Size([32])
Test ValDataLoader (First Fold):
Batch 1:
PadWords shape: torch.Size([32, 54])
PadLabels shape: torch.Size([32, 54])
Mask shape: torch.Size([32, 54])
Lengths shape: torch.Size([32])
Test TestDataLoader:
Batch 1:
PadWords shape: torch.Size([32, 47])
PadLabels shape: torch.Size([32, 47])
Mask shape: torch.Size([32, 47])
Lengths shape: torch.Size([32])
```

- The test set has 340 sentences (20.05%), and the training + validation set has 1356 sentences (79.95%), which is in line with expectations.↵
- The 5-fold cross validation uses about 1085 sentences (80%) for training and about 271 sentences (20%) for validation in each fold, which enhances training diversity and reduces the risk of overfitting.↵
- The stratified split ensures that the label distribution of the test set and each fold of training and validation sets is consistent with the entire dataset and the proportion is balanced, which verifies the effectiveness of the stratified strategy.↵

Neural Networks

2.1 BiLSTM network for NER

2.1.1 Network Overview

- Setting the Embedding layer encodes discrete words into learnable continuous vector representations to capture semantic information.
- A two-layer bidirectional LSTM is used to capture the context information of the sequence and improve the modeling ability of the context. The total dimension of the hidden state is 128, 64 in each direction (128 // 2), batch_first=True supports batch processing of data, and dropout=0.3
- The LSTM output is layer-normalized to stabilize the training process and speed up

convergence, thereby improving the model's generalization ability.

- The dropout ratio is 0.5 to prevent overfitting.
- Finally, the output of BiLSTM is mapped to the tag space (TagSize) to generate the prediction results of each word for subsequent cross entropy calculation.

```
class BiLSTM(nn.Module):
  def __init__(self, WordSize, TagSize, EmbeddingDim, HiddenDim):
    super(BiLSTM, self).__init__()
    self.Embedding = nn.Embedding(WordSize, EmbeddingDim, padding_idx=0)
    self.LSTM = nn.LSTM(EmbeddingDim, HiddenDim // 2,num_layers=2,
                        bidirectional=True, batch_first=True, dropout=0.3)
    self.Dropout = nn.Dropout(0.5)
    self.HiddenTag = nn.Linear(HiddenDim, TagSize)
    self.LayerNorm = nn.LayerNorm(HiddenDim)
```

2.1.2 Component Selection

- **Loss function:** nn.CrossEntropyLoss(ignore_index=0):
  - Sequences often contain padding to align sentences of different lengths. Ignore the padding index to avoid invalid data interfering with the loss calculation.
  - The cross-entropy loss combined with the softmax function can effectively measure the difference between the model prediction distribution and the true label distribution, and is suitable for optimizing classification tasks.

```
def Loss(self, Sentences, Tags, Lengths):
  Emissions = self.forward(Sentences, Lengths)
  Mask = (Sentences != 0).float()
  Loss = nn.CrossEntropyLoss(ignore_index=0)
  return Loss(Emissions.view(-1, Emissions.shape[-1]), Tags.view(-1))
```

- **Optimizer:** torch.optim.Radam(learning_rate=5e-4, weight_decay=8e-4):
  - The learning rate of 5e-4 is a compromise value that ensures that the model learns quickly while avoiding oscillation caused by too high a learning rate.
  - Weight decay 8e-4 is used as L2 regularization to prevent the model from overfitting.

```
Optimizer = torch.optim.RAdam(Model.parameters(), lr=LR, weight_decay=8e-4)
```

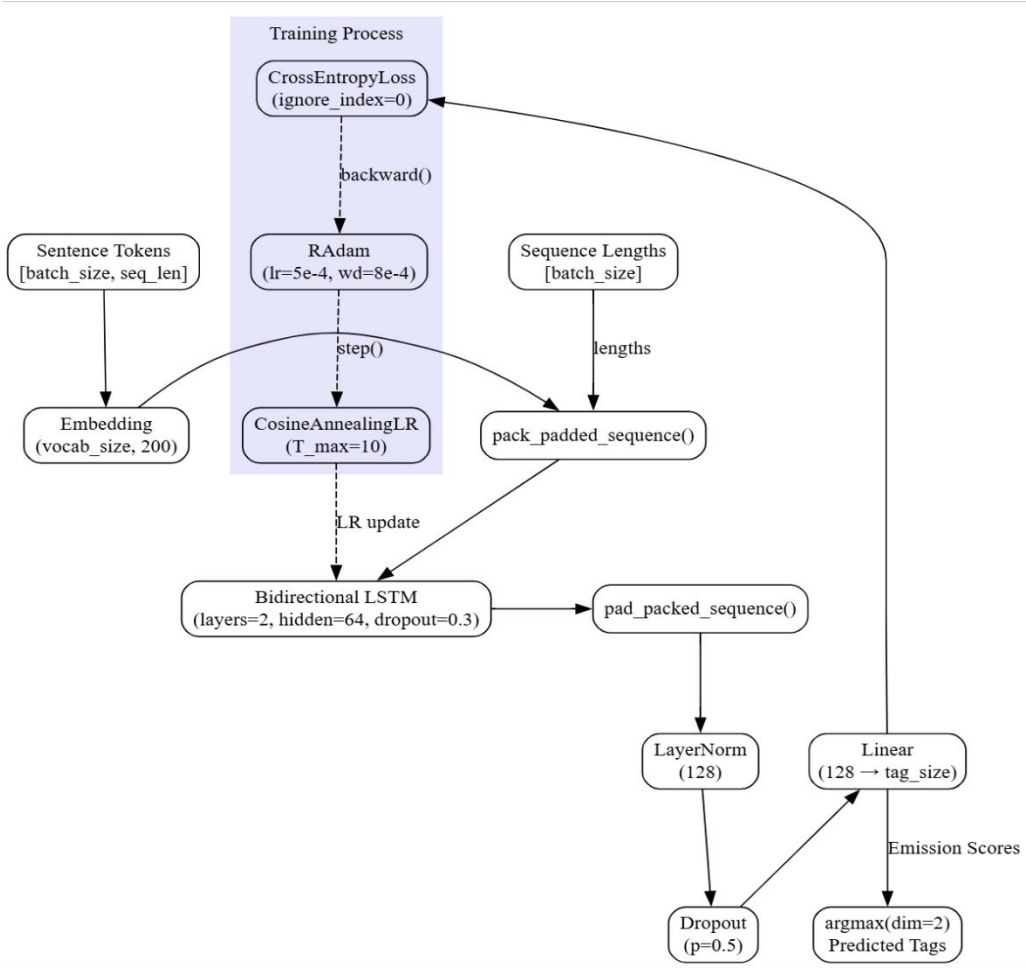
- **Learning Rate Scheduler:** torch.optim.lr_scheduler.CosineAnnealingLR(*T_max*=10, *eta_min*=1e-5):
  - The maximum period of 10 means that the learning rate completes a full cosine annealing cycle every 10 epochs, balancing the exploration of the

learning rate and stable convergence.

- Minimum learning rate of 1e-5 allows fine-tuning in late training without halting learning.

```
Scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(Optimizer, T_max=10, eta_min=1e-5)
```

- **Regularization Techniques:**
  - Dropout: 30% Dropout within the LSTM and 50% after the output to prevent overfitting. The higher 50% rate regularizes the output layer effectively, especially with limited data, while the 30% rate in the LSTM balances learning ability and overfitting prevention.
  - Gradient Clipping: Use torch.nn.utils.clip_grad_norm_ with a max norm of 1.0 to prevent gradient explosion, particularly for long sequences or noisy data.
  - Layer Normalization: Apply nn.LayerNorm after the LSTM output to normalize hidden states, reducing internal covariate shift, stabilizing training, and speeding up convergence. Unlike batch normalization, it suits RNNs as it doesn't rely on batch statistics and handles variable-length sequences.
- **Flow chart of BiLSTM network:**



1.4 Transformer Network

1.4.1 Network Overview

- **Embedding Layer:** Use nn.Embedding to map word indices to a 128-dimensional embedding vector space and use Xavier uniform initialization to initialize parameters to ensure initial training stability.
- **Positional Encoding:** The model introduces fixed sine-cosine positional encoding, which supports a maximum sequence length of 150. It is calculated and registered as a trainable parameter (nn.Parameter) through the init_positional_encoding method to preserve word order information.

- **Transformer Encoder:** Use nn.TransformerEncoderLayer to construct a Transformer encoder layer and set the following parameters:
  - The number of multi-head attention heads is 4;
  - The feedforward neural network dimension is 4 times the embedding dimension, that is, 512;
  - The activation function is ReLU;
  - Enable batch_first=True;
  - The dropout probability is 0.5 to prevent overfitting.

```
class TransformerNER(nn.Module):
    def __init__(self, WordSize, NumTag, EmbeddingDim=128, NumHeads=4, NumLayers=1, DropOut=0.5, Maxlen=150):
        super(TransformerNER, self).__init__()
        self.Embedding = nn.Embedding(WordSize, EmbeddingDim)
        self.EmbeddingDim = EmbeddingDim
        self.PositionalEncoding = nn.Parameter(torch.zeros(1, Maxlen, EmbeddingDim))
        self.init_positional_encoding(Maxlen, EmbeddingDim)
        EncoderLayer = nn.TransformerEncoderLayer(
            d_model=EmbeddingDim,
            nhead=NumHeads,
            dim_feedforward=EmbeddingDim * 4,
            dropout=DropOut,
            activation='relu',
            batch_first=True
        )
        self.TransformerEncoder = nn.TransformerEncoder(EncoderLayer, NumLayers)
        self.fc = nn.Linear(EmbeddingDim, NumTag)
        self.dropout = nn.Dropout(DropOut)
        self._init_weights()
```

- **Fully connected layer (Linear)：** The final nn.Linear layer maps the features output by the Transformer to the label space (NER category) to predict the label of each word. The weights of this layer are still initialized using Xavier, and the bias is set to 0.

- **Masking Mechanism:** Generate a mask for ignoring padding positions through the CreatMask function so that the model does not consider these positions when

calculating losses and predictions

- **Forward process:**
  - The input is a batch of word indexes and the actual length of each sequence;
  - After adding embedding and position encoding, it is passed to the Transformer encoder;
  - The output result is the final word-level label prediction result after passing through the fully connected layer.

```
def forward(self, SRC, Lengths):
    BatchSize, Seqlen = SRC.size()

    SRCEmbedding = self.Embedding(SRC) * math.sqrt(self.EmbeddingDim)
    PEncoding = self.PositionalEncoding[:, :Seqlen, :].expand(BatchSize, -1, -1)
    SRCEmbedding = SRCEmbedding + PEncoding.to(SRCEmbedding.device)
    SRCEmbedding = self.dropout(SRCEmbedding)

    Mask = self.CreatMask(SRC, Lengths)

    Output = self.TransformerEncoder(SRCEmbedding, src_key_padding_mask=~Mask)
    Output = self.dropout(Output)
    Output = self.fc(Output)
```

1.4.2 Component Selection

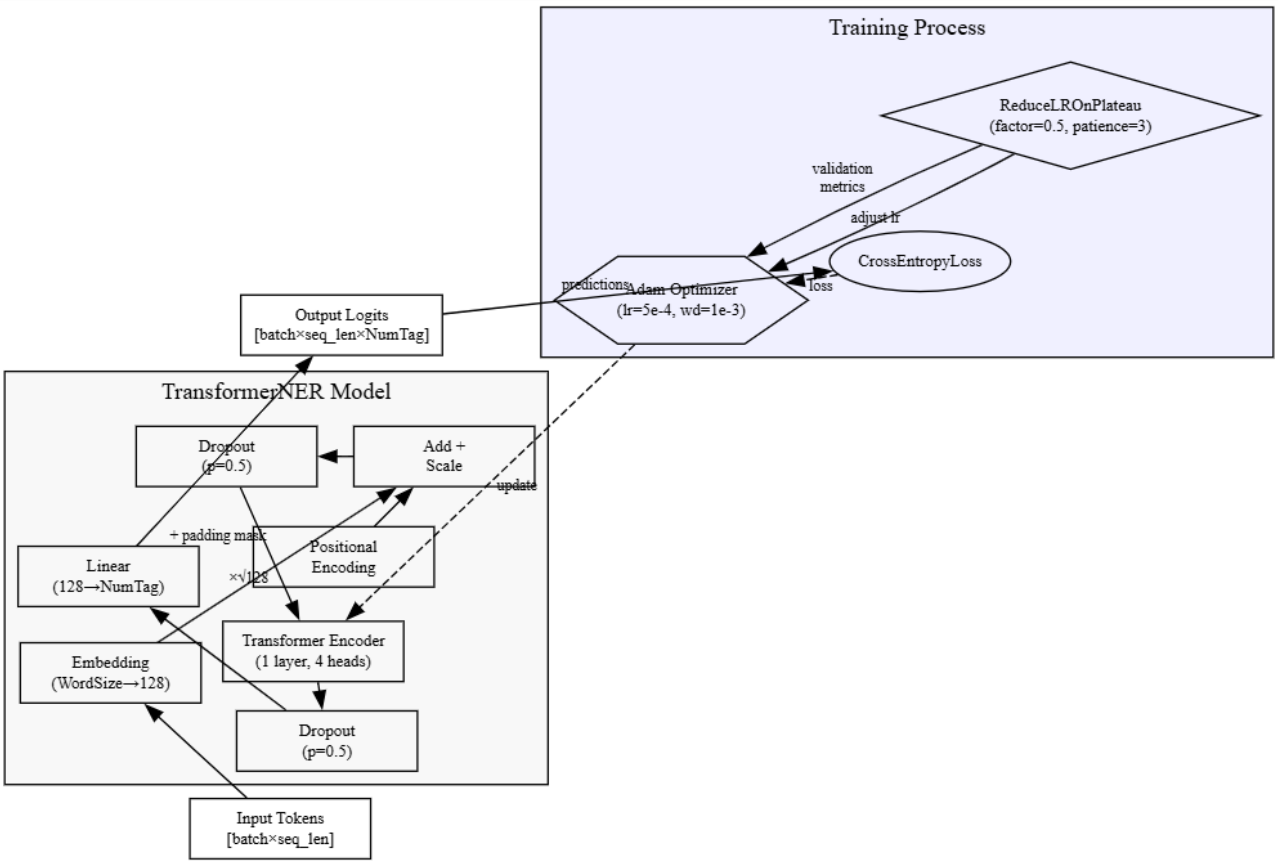
- **Loss Function:** Use nn.CrossEntropyLoss(reduction='none') to get the loss value of each word position. Since there are padding positions, use the custom mask ActiveLoss to filter out these positions and only calculate the average loss for valid words. This processing method can avoid the interference of padding on model training.

- **Optimizer:** Adam and the learning rate is set to 5e-4, and the weight decay is set to 1e-3 for regularization.

- **Learning rate scheduler:** Use ReduceLROnPlateau learning rate scheduler to automatically halve the learning rate after the validation loss stops decreasing. Patience=3 means that the learning rate will be adjusted only after three consecutive cycles without improvement. This helps to further fine-tune the training process and prevent the model from oscillating or falling into a local optimum.

- **Dropout regularization:** Dropout layers are added to multiple key positions in the model (after embedding and after Transformer encoding), with a dropout probability of 0.5, which effectively prevents overfitting and improves generalization ability, especially on smaller training sets.

- **Flow chart of BiLSTM network:**



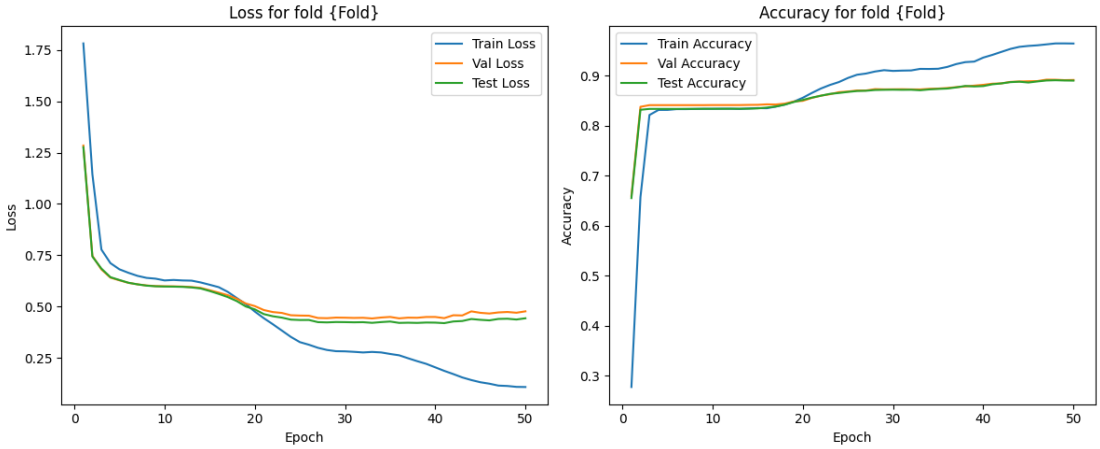
1.5 Report table of the results

Model	Fold	Train Loss	Val Loss	Test Loss	Train Acc	Val Acc	Test Acc
BiLSTM	1	0.1049	0.4067	0.437	0.9658	0.8912	0.8828
	2	0.1085	0.5179	0.4486	0.9657	0.878	0.8906
	3	0.1065	0.4403	0.4488	0.9656	0.8899	0.8893
	4	0.1014	0.4714	0.4472	0.9665	0.8889	0.889
	5	0.1093	0.4776	0.4438	0.9641	0.8912	0.8899
Average	—	**0.1061**	**0.4628**	**0.4451**	**0.9655**	**0.8878**	**0.8883**

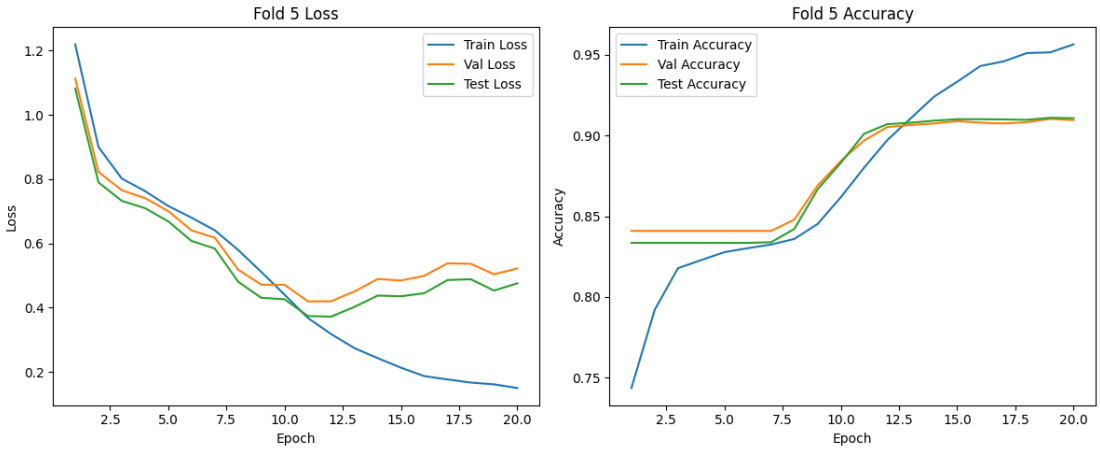
Fold	Train Loss	Val Loss	Test Loss	Train Acc	Val Acc	Test Acc
1	0.174	0.3995	0.4409	0.9481	0.914	0.9072
2	0.136	0.5327	0.4784	0.9604	0.9001	0.9081
3	0.1691	0.4874	0.4417	0.9501	0.9047	0.9114
4	0.1534	0.4466	0.4539	0.9557	0.913	0.9071
5	0.1716	0.5067	0.4475	0.9497	0.91	0.9095
Average	**0.1608**	**0.4746**	**0.4525**	**0.9528**	**0.9084**	**0.9087**

In this experiment, both BiLSTM and Transformer models were evaluated using 5-fold cross-validation for the NER task. The Transformer consistently outperformed BiLSTM in terms of both validation accuracy (90.84% vs. 88.78%) and test accuracy (90.87% vs. 88.83%). Furthermore, it achieved a lower average test loss (0.4525 vs. 0.4451), indicating better generalization.

Although the Transformer had slightly higher training loss, this is likely due to its greater model complexity. Overall, the Transformer showed better and more stable performance, and is therefore considered the best-performing model in this experiment.↵



BiLSTM



Transformer

Overfitting occurs because the model becomes too closely fitted to the training data, including memorizing noise or irregular patterns. As a result, the model performs well on the training set but fails to generalize to unseen data like the validation or test sets. This is especially common in models with high capacity, such as the Transformer. Secondly, this phenomenon is particularly obvious when the data set is too small or the label distribution is extremely uneven.↵

From the training curves, it is clear that both models — BiLSTM and Transformer — exhibit signs of overfitting as training progresses. Specifically, while the training loss continues to decrease and training accuracy increases, the validation loss begins to rise or plateau after a certain number of epochs, indicating that the model's performance on unseen data starts to degrade.↵

This pattern suggests that both models are learning the training data too well, including its noise or outliers, which leads to a decline in generalization. The widening gap between training and validation loss is a typical symptom of overfitting.

2. Evaluation

1. **Test Loss**:↵
    - **Value**: 0.4409↵
    - **Explanation**: The loss function measures the difference between the model's predictions and the true labels. A lower loss indicates a better fit to the data. The current value suggests that the model's predictions on the test set are stable, with no clear signs of overfitting or underfitting.↵
2. **Accuracy**:↵
    - **Value**: 0.9072↵
    - **Explanation**: Accuracy reflects the proportion of correctly predicted labels. An accuracy above 90% indicates that the model correctly identifies entity classes at most positions, demonstrating strong overall performance.↵
3. **Precision (weighted)**:↵
    - **Value**: 0.8982↵
    - **Explanation**: Precision measures how many of the samples predicted as a certain class truly belong to that class. In the NER task, high precision indicates that the model rarely mislabels non-entity tokens as entities.↵
4. **Recall (weighted)**:↵
    - **Value**: 0.9072↵

- **Explanation**: Recall reflects how many of the actual instances of a class are correctly identified by the model. The recall being close to accuracy suggests that the model misses few actual entities, showing good coverage.↵

5. **F1-Score (weighted)**:↵
   - **Value**: 0.8961↵
   - **Explanation**: The F1-score is the harmonic mean of precision and recall, serving as a balanced metric especially useful when dealing with class imbalance. A value close to 0.9 confirms that the model performs robustly across all classes.↵

In summary, the TransformerNER model performs well in all indicators on the test set, has strong generalization ability and prediction accuracy, and can meet the requirements of named entity recognition tasks for accuracy and stability.

Confusion matrix:

From the matrix, we can observe that the model performs best when recognizing the non-entity class (O), while it struggles more with distinguishing between the various entity types.

- **Non-Entity (O):**
  - Correct predictions: 6507
  - The model demonstrates very high accuracy in identifying non-entity tokens, indicating a stable ability to recognize regular, non-named tokens.
  - Misclassification into entity classes (especially I-ORG and I-MISC) is relatively rare, suggesting that the model takes a conservative approach, often predicting tokens as "O" when uncertain.

- **Location (I-LOC):**
  - Correct predictions: 183
  - Misclassified as O: 96, which is more than 30% of all I-LOC instances.
  - This indicates that the model has insufficient grasp of location boundary or semantic features, making it prone to false negatives.

- **Miscellaneous (I-MISC):**
  - Correct predictions: 101
  - Misclassified as O: 160
  - The model finds this category particularly difficult, likely due to its vagueness and broad semantic coverage.

- **Organization (I-ORG):**
  - Correct predictions: 202
  - Misclassified as O: 135
  - Also misclassified into I-LOC and I-MISC, showing the model struggles to distinguish between organization names and other entity types with similar context.

- **Person (I-PER):**
  - Correct predictions: 161
  - Misclassified as O: 142
  - Similar to I-ORG, suggesting the model has trouble distinguishing person names from surrounding nouns or verbs, leading to false negatives.

The Transformer-based NER model demonstrates strong performance in identifying non-entity tokens (O), indicating its robustness and resistance to overfitting in general word recognition. However, it exhibits low recall for named entity classes, with a tendency to misclassify them as non-entities, especially for categories like I-MISC and I-LOC. Furthermore, there is notable confusion among similar entity types such as I-ORG, I-LOC, and I-MISC, suggesting limitations in semantic and boundary modeling. To address these issues, future improvements could include integrating sequential models like BiLSTM or CRF to enhance boundary recognition, leveraging richer contextual representations, and applying techniques like data augmentation or class-weighted loss functions to mitigate class imbalance and improve overall entity classification accuracy.