NER Task Using T-NER Model: A Replication

Muhammad Hanif Fahreza*[§] and Taufiq Hadi Pratama[†][§]
Faculty of Computer Science
Universitas Indonesia
Depok, Indonesia
Email: *taufiq.hadi@ui.ac.id, [†]muhammad.hanif94@ui.ac.id

***Abstract*—This paper provides an overview of finetuning pre-trained models for NER task using T-NER, a Python framework for assisting researchers in finetuning NER pre-trained models. We replicated the experiment that is done by the creators of T-NER in an environment with less resources. The experiment was evaluated using commonly used metrics such as precision, recall, and F1 score. The result shows only a slight decrease of performance in the replicate experiment despite a huge gap of resources used. Overall, the examined framework provides a promising tool for assisting researchers in finetuning pre-trained models for NER task.**

***Index Terms*—NER, BERT, Language Model**

I. Introduction

The NER task, also known as named entity recognition, is a fundamental problem in natural language processing. It involves identifying and classifying named entities, such as people, organizations, locations, and dates, in unstructured text. This is an important step in many downstream NLP tasks, such as information extraction, machine translation, and text summarization.

The goal of the reference paper is contribute with creating a framework that makes fine-tuning NER with Transformers models easier. With this contribution, everyone can reproduce a scenario or just fine-tuning the pre-trained model for their specific task [1]. While in this paper, we reproduce the experiment conducted in the paper with reduced scenarios such as smaller datasets, lesser training epochs, and simpler hyperparameter configuration.

Section II describes about related methods that build up this paper, such as Deep Learning, NER task, and language model. To understand the framework, we explain step-by-step of the framework workflow in section III by conducting mini experiment and display each result of the step. Experiments description and results reproduced from the main paper can be seen in section IV. After the experiment, we conclude the inferred information from the results in section V while also giving suggestions for next research.

II. Related Works

A. Named Entity Recognition (NER)

Named Entity Recognition is the process of analyzing a sentence or part of text to find entities that can be classified into categories such as name, organization, location, quantity, amount, geolocation, etc. Traditional NER algorithm usually only includes name, location, and organization [2]. However, it can now be dynamically trained to extract more than just the aforementioned entities. In short, given an arbitrary text, NER's job is to recognize named entities and identify their types. For example, given the sentence "Sukarno proclaimed the independence of the Republic of Indonesia on 1945.", the NER model identifies "Sukarno" as a person and "Republic of Indonesia" as an organization.

Traditionally, NER systems have relied on classification models based on hand-constructed feature sets extracted from corpora. Sari et al. in [3] utilized Stanford POS Tagger to identify part-of-speeches that usually corresponds with entities within a sentence. For example, location entities are exclusively categorized as noun in a sentence. After that, rule representation from RAPIER is used in a self-training algorithm to label each token according to its corresponding entities. However, the current state-of-the-art of NER is to exploit deep learning architectures.

B. Language Model

A language model is a probability distribution over a word sequence [4]. For a word sequence with m words, the language model assigns a probability $P(w_1, w_2, ..., w_m)$ across the sequence. A language model generates probabilities by training a text corpus in one or more languages. Because a language can be used to represent an infinite variety of valid sentences, language modeling tries to add non-zero probabilities to linguistically valid sequences that may never exist in the training data.

Because the category of named entity is highly dependent on the textual semantics and its surrounding context, language model can be used for NER task to model the probability distribution of the token given its surrounding context [5]. The text representation that is generated by the language model can be transferred on a pipelined task such as NER. Before the age of neural networks, probabilistic models such as n-gram models using Markov assumption are often applied as the language model in NER tasks.

However, the advent of recurrent neural networks (RNN) in NLP leads them to become a better choice of coding layers for language model such as Long Short Term Memory (LSTM) and Bidirectional Encoder Representations from Transformers (BERT) are still used. The most successful approach to language models is to pre-train the language model on common texts to predict the next sentence. This linguistic knowledge

[§]These authors contributed equally to this work.

is transferred by fitting the model to the target task. The tasks covered range from sentiment analysis in government policies, question answering, to recognizing named entities in sentences.

C. NER with Deep Learning

The current state-of-the-art of NER with deep learning is by leveraging BERT as both the language model and word embedding. BERT architecture has the ability to see the context embedded in an input in two directions so that it has the ability to understand the sentence as a whole [6]. BERT uses various alternatives as needed because the results of the model depending on the what stage it is currently in and what kind of data is used for training. One of these BERT alternatives can be applied for sequence labelling tasks.

Sequence labeling tasks such as part-of-speech tagging or BIO-based named entity recognition follow the same basic classification approach [4]. The final output vector corresponds to each input token is passed to the classifier in order to create a distribution over the set of possible entities (e.g. softmax distribution). Assuming a simple classifier consisting of a single feedforward layer followed by softmax, we need to learn a set of weights for this additional layer. We can use a greedy approach that utilizes each token's argmax tag as an input for feedforward layer to generate the final output tag sequence. Another way to get the weights for the additional layer is by using a pre-trained model's weights and transfer it to our model.

D. T-NER

T-NER is an open-source Python framework to assist fine-tuning a deep learning model for NER task. The flowchart of how T-NER works is given in figure 1.

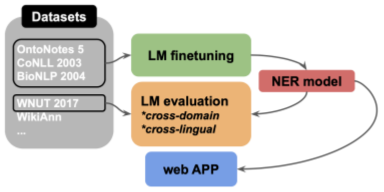


Fig. 1. Example of code to tokenize a given dataset

We can customize the input dataset for the NER task by choosing our own custom dataset or by using existing dataset that is already provided by the framework. However, we cannot fine-tune a custom pre-trained model using T-NER since the framework only allows using models from HuggingFace[1]. After the dataset and the model is prepared, we can start the fine-tuning process for our specific task. The resulting fine-tuned model can be applied for predicting test dataset and further evaluation.

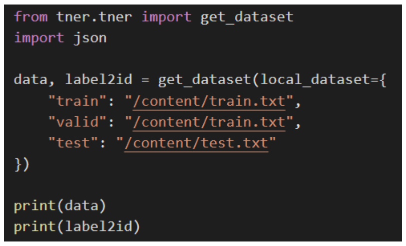
[1]https://huggingface.co/

III. Program Workflow

In this section, we describe the workflow of T-NER framework in training its language model for NER task. This includes tasks such as processing the raw input texts to its embedding vector until how it predicts the named-entities for a sentence.

A. Tokenization

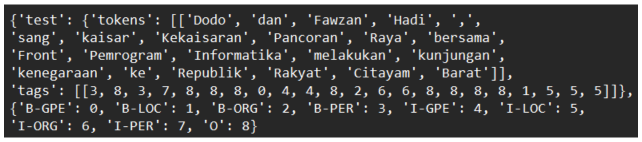
The T-NER framework provides a method to automatically tokenize and assign ID to the named-entity labels in a text as in figure 2.



```
from tner.tner import get_dataset
import json

data, label2id = get_dataset(local_dataset={
    "train": "/content/train.txt",
    "valid": "/content/train.txt",
    "test": "/content/test.txt"
})

print(data)
print(label2id)
```

Fig. 2. Example of code to tokenize a given dataset

The result of the code above can be seen in figure 3.



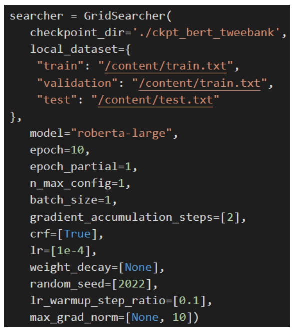
```
{'test': {'tokens': [['Dodo', 'dan', 'Fawzan', 'Hadi', ',',
'sang', 'kaisar', 'Kekaisaran', 'Pancoran', 'Raya', 'bersama',
'Front', 'Pemrogram', 'Informatika', 'melakukan', 'kunjungan',
'kenegaraan', 'ke', 'Republik', 'Rakyat', 'Citayam', 'Barat']],
'tags': [[3, 8, 3, 7, 8, 8, 8, 0, 4, 4, 8, 2, 6, 6, 8, 8, 8, 8, 1, 5, 5, 5]]},
{'B-GPE': 0, 'B-LOC': 1, 'B-ORG': 2, 'B-PER': 3, 'I-GPE': 4, 'I-LOC': 5,
'I-ORG': 6, 'I-PER': 7, 'O': 8}
```

Fig. 3. Result of tokenization using T-NER framework for test dataset

As we can see, the result of the tokenization method is the tokens that is already extracted from the given dataset. It also assigns IDs to each found named-entity labels from the dataset. The tag for each tokens can also be found in the tags property in which each token is represented by its assigned ID.

B. Grid Search

T-NER framework also provides a class to employ grid search method in hyperparameter tuning for a given language model in conducting NER task. The configuration of the class initialization can be seen in the figure 4



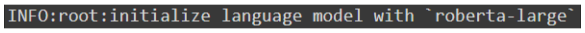
```
searcher = GridSearcher(
   checkpoint_dir='./ckpt_bert_tweebank',
   local_dataset={
    "train": "/content/train.txt",
    "validation": "/content/train.txt",
    "test": "/content/test.txt"
},
   model="roberta-large",
   epoch=10,
   epoch_partial=1,
   n_max_config=1,
   batch_size=1,
   gradient_accumulation_steps=[2],
   crf=[True],
   lr=[1e-4],
   weight_decay=[None],
   random_seed=[2022],
   lr_warmup_step_ratio=[0.1],
   max_grad_norm=[None, 10])
```

Fig. 4. T-NER's Grid Search class initialization

We only provide one value for each hyperparameter except max_grad_norm in this example, which is equivalent of almost not using grid search method due to limited resources. We use pre-trained roberta-large model as the language model and fine tune it in 10 epochs using our dummy dataset which consists of five sentences.

C. Training Process

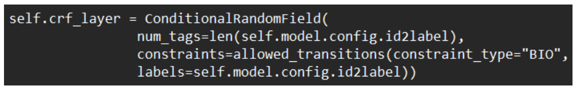
1) Initiate The Language Model
We choose Roberta language model for this demonstration as shown in figure 5.



```
INFO:root:initialize language model with `roberta-large`
```

Fig. 5. Roberta-Large language model initialization

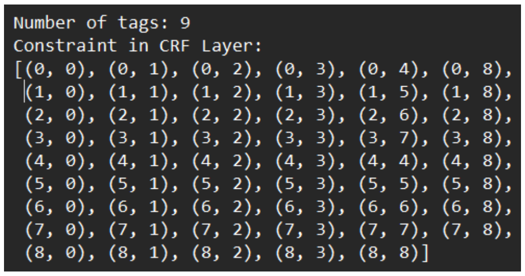
2) Initialize CRF instance
We initialize a CRF layer to be embedded in front of the softmax output layer. The CRF layer functions as a filter for the prediction made by the softmax output layer so that the prediction is within the given CRF constraints. The CRF's hyperparameter can be seen in figure 6



```
self.crf_layer = ConditionalRandomField(
                num_tags=len(self.model.config.id2label),
                constraints=allowed_transitions(constraint_type="BIO",
                labels=self.model.config.id2label))
```

Fig. 6. CRF Initialization

The value for the hyperparameter num_tags and constraints can be seen in figure 7

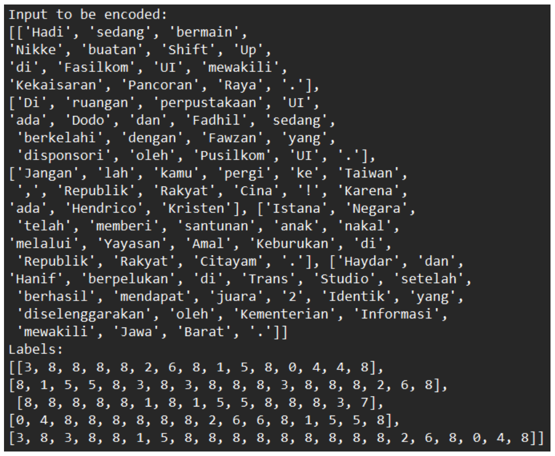


```
Number of tags: 9
Constraint in CRF Layer:
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 8),
 (1, 0), (1, 1), (1, 2), (1, 3), (1, 5), (1, 8),
 (2, 0), (2, 1), (2, 2), (2, 3), (2, 6), (2, 8),
 (3, 0), (3, 1), (3, 2), (3, 3), (3, 7), (3, 8),
 (4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (4, 8),
 (5, 0), (5, 1), (5, 2), (5, 3), (5, 5), (5, 8),
 (6, 0), (6, 1), (6, 2), (6, 3), (6, 6), (6, 8),
 (7, 0), (7, 1), (7, 2), (7, 3), (7, 7), (7, 8),
 (8, 0), (8, 1), (8, 2), (8, 3), (8, 8)]
```

Fig. 7. CRF hyperparameters

The list of tuples that is used as the CRF constraint represents the allowed transitions of a tag to another tag. For example, the tuple (0, 1) means that a word with tag 0 can be followed by a word with tag 1. This constraint will be used to filter the prediction of the output softmax layer.

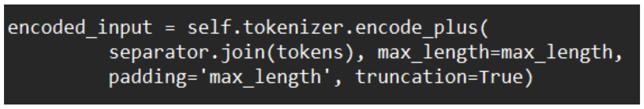
3) Tokenize Input Dataset
Now, the training process begins. The input text will be tokenized and its named-entities will be assigned with IDs in a similar process as in subsection III-A. The result of the tokenization can be seen in figure 8.



```
Input to be encoded:
[['Hadi', 'sedang', 'bermain',
'Nikke', 'buatan', 'Shift', 'Up',
'di', 'Fasilkom', 'UI', 'mewakili',
'Kekaisaran', 'Pancoran', 'Raya', '.'],
['Di', 'ruangan', 'perpustakaan', 'UI',
'ada', 'Dodo', 'dan', 'Fadhil', 'sedang',
 'berkelahi', 'dengan', 'Fawzan', 'yang',
 'disponsori', 'oleh', 'Pusilkom', 'UI', '.'],
['Jangan', 'lah', 'kamu', 'pergi', 'ke', 'Taiwan',
 ',', 'Republik', 'Rakyat', 'Cina', '!', 'Karena',
'ada', 'Hendrico', 'Kristen'], ['Istana', 'Negara',
 'telah', 'memberi', 'santunan', 'anak', 'nakal',
'melalui', 'Yayasan', 'Amal', 'Keburukan', 'di',
 'Republik', 'Rakyat', 'Citayam', '.'], ['Haydar', 'dan',
'Hanif', 'berpelukan', 'di', 'Trans', 'Studio', 'setelah',
 'berhasil', 'mendapat', 'juara', '2', 'Identik', 'yang',
 'diselenggarakan', 'oleh', 'Kementerian', 'Informasi',
 'mewakili', 'Jawa', 'Barat', '.']]
Labels:
[[3, 8, 8, 8, 8, 2, 6, 8, 1, 5, 8, 0, 4, 4, 8],
[8, 1, 5, 5, 8, 3, 8, 3, 8, 8, 8, 3, 8, 8, 8, 2, 6, 8],
 [8, 8, 8, 8, 8, 1, 8, 1, 5, 5, 8, 8, 8, 3, 7],
[0, 4, 8, 8, 8, 8, 8, 8, 2, 6, 6, 8, 1, 5, 5, 8],
[3, 8, 3, 8, 8, 1, 5, 8, 8, 8, 8, 8, 8, 8, 8, 8, 2, 6, 8, 0, 4, 8]]
```

Fig. 8. Dummy dataset input for training

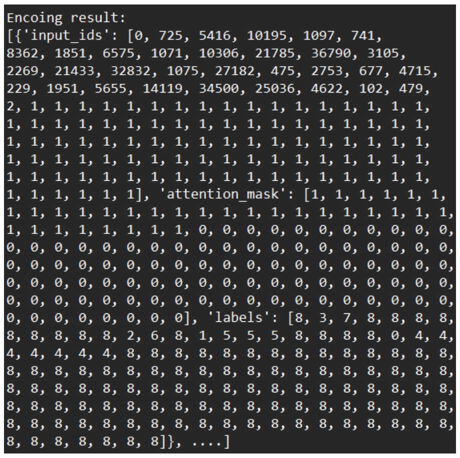
4) Encode Input Dataset
After the tokenization process is complete, the resulting tokens will be converted to a numerical form. In this demonstration, this process is done by the Roberta-large language model using its encode_plus method as shown in figure 9.



```
encoded_input = self.tokenizer.encode_plus(
        separator.join(tokens), max_length=max_length,
        padding='max_length', truncation=True)
```

Fig. 9. Roberta token encoding method

The example of an encoded sentence can be seen in figure 10.



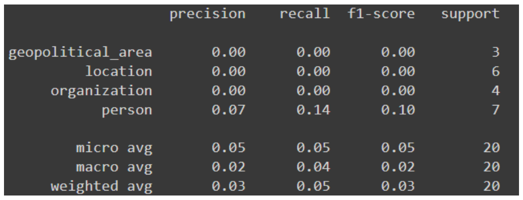
```
Encoing result:
[{'input_ids': [0, 725, 5416, 10195, 1097, 741,
8362, 1851, 6575, 1071, 10306, 21785, 36790, 3105,
2269, 21433, 32832, 1075, 27182, 475, 2753, 677, 4715,
229, 1951, 5655, 14119, 34500, 25036, 4622, 102, 479,
2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1], 'attention_mask': [1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0], 'labels': [8, 3, 7, 8, 8, 8, 8,
8, 8, 8, 8, 8, 2, 6, 8, 1, 5, 5, 5, 8, 8, 8, 8, 0, 4, 4,
4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
8, 8, 8, 8, 8, 8, 8]}, ....]
```

Fig. 10. Encoded input example

The property input_ids represents the IDs of a sentence's tokens. It always begins with ID 0 since it represents the special [CLS] token that indicates the beginning of a text. The ID 2 represents the special [SEP] token which separates every sentence in the tokens. Since our dummy dataset only contains texts that consist only of one sentence, the [SEP] token represents the end of the text. Series of ID 1 after the [SEP] token represents meaningless padding tokens to extend the text such that it has the same length as the longest text in the dataset. This is needed in order to represent the whole input dataset as a tensor.

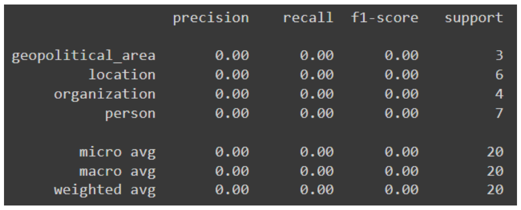
The attention_mask property indicates which tokens in the input_ids should be given attention and which input_ids that is a padding token. 1 represents real tokens and 0 represents padding tokens. The labels property represents the named-entity ID label of each input_ids.

5) Initialize Training Weight
As shown in figure 4, we set the epoch_partial value to 1. This means the initial random weight will be trained in 1 partial epoch before the real training is conducted. This is done so that the initial weight of the real training process is not as random as without this initial training. This initial training process is also used to select the best hyperparameter configuration given to the grid search. As shown in figure 4, we only give two possible hyperparameter configurations: One with hyperparameter max_grad_norm = None and one with max_grad_norm = 10.



```
                   precision    recall  f1-score   support

geopolitical_area       0.00      0.00      0.00         3
         location       0.00      0.00      0.00         6
     organization       0.00      0.00      0.00         4
           person       0.07      0.14      0.10         7

        micro avg       0.05      0.05      0.05        20
        macro avg       0.02      0.04      0.02        20
     weighted avg       0.03      0.05      0.03        20
```

Fig. 11. NER task result using weight given by the first configuraton



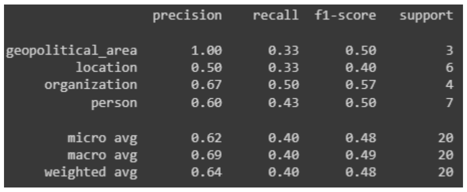
```
                   precision    recall  f1-score   support

geopolitical_area       0.00      0.00      0.00         3
         location       0.00      0.00      0.00         6
     organization       0.00      0.00      0.00         4
           person       0.00      0.00      0.00         7

        micro avg       0.00      0.00      0.00        20
        macro avg       0.00      0.00      0.00        20
     weighted avg       0.00      0.00      0.00        20
```

Fig. 12. NER task result using weight given by the second configuraton

As seen from figure 11 and figure 12, the configuration with hyperparameter max_grad_norm = None performs better than the latter one. Thus, the real training process will use the weight given by the initial training weight with the first configuration.

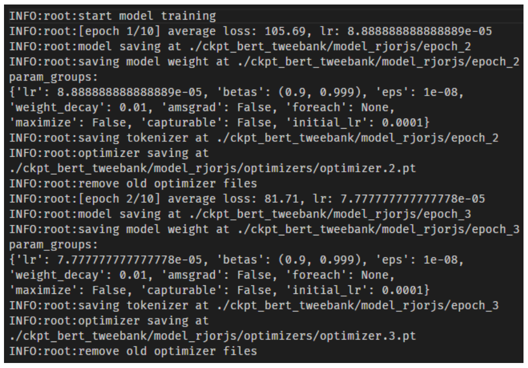
6) Training Process
Now we move on to the real training process. We set the training process to be done in 10 epochs as shown in figure 4. The weight of the NER model will be updated in each epoch according to the computed loss. Beside of the weight, the learning rate will also be updated by the optimizer (in this case AdamW). T-NER keeps the record of each epoch's model. When the training process is over, T-NER will select the model with the best result from the record.

The training process from one epoch to the other, which also includes learning rate update by the optimizer can be seen in figure 14. The confusion matrix resulted by the best model is in figure 13.



```
                   precision    recall  f1-score   support

geopolitical_area       1.00      0.33      0.50         3
         location       0.50      0.33      0.40         6
     organization       0.67      0.50      0.57         4
           person       0.60      0.43      0.50         7

        micro avg       0.62      0.40      0.48        20
        macro avg       0.69      0.40      0.49        20
     weighted avg       0.64      0.40      0.48        20
```

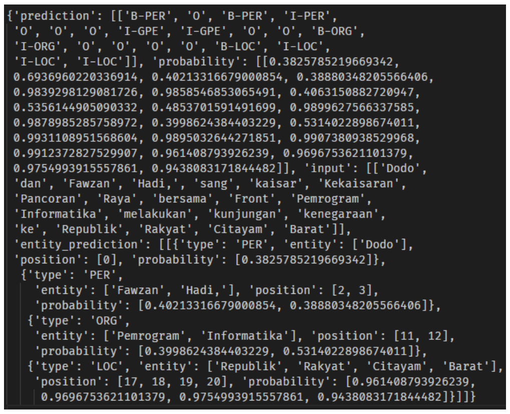
Fig. 13. Confusion matrix by the best model



```
INFO:root:start model training
INFO:root:[epoch 1/10] average loss: 105.69, lr: 8.888888888888889e-05
INFO:root:model saving at ./ckpt_bert_tweebank/model_rjorjs/epoch_2
INFO:root:saving model weight at ./ckpt_bert_tweebank/model_rjorjs/epoch_2
param_groups:
{'lr': 8.888888888888889e-05, 'betas': (0.9, 0.999), 'eps': 1e-08,
'weight_decay': 0.01, 'amsgrad': False, 'foreach': None,
'maximize': False, 'capturable': False, 'initial_lr': 0.0001}
INFO:root:saving tokenizer at ./ckpt_bert_tweebank/model_rjorjs/epoch_2
INFO:root:optimizer saving at
./ckpt_bert_tweebank/model_rjorjs/optimizers/optimizer.2.pt
INFO:root:remove old optimizer files
INFO:root:[epoch 2/10] average loss: 81.71, lr: 7.777777777777778e-05
INFO:root:model saving at ./ckpt_bert_tweebank/model_rjorjs/epoch_3
INFO:root:saving model weight at ./ckpt_bert_tweebank/model_rjorjs/epoch_3
param_groups:
{'lr': 7.777777777777778e-05, 'betas': (0.9, 0.999), 'eps': 1e-08,
'weight_decay': 0.01, 'amsgrad': False, 'foreach': None,
'maximize': False, 'capturable': False, 'initial_lr': 0.0001}
INFO:root:saving tokenizer at ./ckpt_bert_tweebank/model_rjorjs/epoch_3
INFO:root:optimizer saving at
./ckpt_bert_tweebank/model_rjorjs/optimizers/optimizer.3.pt
INFO:root:remove old optimizer files
```

Fig. 14. Epoch 1 and 2 run details

7) Predict Test Dataset Labels
The prediction result of the dummy test dataset from the best model can be seen in figure 15.



```
{'prediction': [['B-PER', 'O', 'B-PER', 'I-PER',
 'O', 'O', 'O', 'I-GPE', 'I-GPE', 'O', 'O', 'B-ORG',
 'I-ORG', 'O', 'O', 'O', 'O', 'B-LOC', 'I-LOC',
 'I-LOC', 'I-LOC']], 'probability': [[0.3825785219669342,
 0.6936960220336914, 0.40213316679000854, 0.38880348205566406,
 0.9839298129081726, 0.9858546853065491, 0.4063150882720947,
 0.5356144905090332, 0.4853701591491699, 0.9899627566337585,
 0.9878985285758972, 0.3998624384403229, 0.5314022898674011,
 0.9931108951568604, 0.9895032644271851, 0.9907380938529968,
 0.9912372827529907, 0.961408793926239, 0.9696753621101379,
 0.9754993915557861, 0.9438083171844482]], 'input': [['Dodo',
 'dan', 'Fawzan', 'Hadi,', 'sang', 'kaisar', 'Kekaisaran',
 'Pancoran', 'Raya', 'bersama', 'Front', 'Pemrogram',
 'Informatika', 'melakukan', 'kunjungan', 'kenegaraan',
 'ke', 'Republik', 'Rakyat', 'Citayam', 'Barat']],
 'entity_prediction': [[{'type': 'PER', 'entity': ['Dodo'],
 'position': [0], 'probability': [0.3825785219669342]},
  {'type': 'PER',
    'entity': ['Fawzan', 'Hadi,'], 'position': [2, 3],
    'probability': [0.40213316679000854, 0.38880348205566406]},
   {'type': 'ORG',
    'entity': ['Pemrogram', 'Informatika'], 'position': [11, 12],
    'probability': [0.3998624384403229, 0.5314022898674011]},
   {'type': 'LOC', 'entity': ['Republik', 'Rakyat', 'Citayam', 'Barat'],
    'position': [17, 18, 19, 20], 'probability': [0.961408793926239,
     0.9696753621101379, 0.9754993915557861, 0.9438083171844482]}]]}
```

Fig. 15. Dummy test dataset prediction result

For each token in the dataset, the model computes each tag probability for the token. The model will select the tag with the highest probability as the label for the token. The labels and the probability of each label assignment can be seen in entity and probability property in figure 15. The tag O is omitted in the result for clarity.

IV. Experiments and Results

In this section, we describe how the experiments will be held, including the dataset description, metrics used to evaluate the results, and the experiment environment. We also give our interpretation on the results of the experiment.

A. Evaluation Metrics

We utilize precision, recall, and F1-score as our evaluation metrics for the resulting prediction of the trained model on the test dataset. Formula for computing those metrics can be seen in equation (1), (2), (3) respectively.

$$\text{precision} = \frac{TruePositive}{TruePositive + FalsePositive} \tag{1}$$

$$\text{recall} = \frac{TruePositive}{TruePositive + FalseNegative} \tag{2}$$

$$\text{f1-score} = \frac{2 \times precision \times recall}{precision + recall} \tag{3}$$

Precision is the proportion of predicted entities that are actually the right entities, while recall is the proportion of actual entities that are predicted as the right entities. The F1 score is the harmonic mean of precision and recall, and is often used as a single metric to evaluate the overall performance of the NER task. We use these metrics to evaluate the experiment because these metrics are often employed for assessing the accuracy and reliability of a NER model.

We compute the micro-average, macro-average, and weighted average of the employed metrics on the whole test dataset. We also compute the metrics on each individual output entity class to better understand the effectiveness of our model for each class independently. This metrics configuration is similar with our main reference paper [1].

B. Dataset

Two datasets will be used from the paper, which are Broad Twitter Corpus (BTC) [7] and Tweebank [8]. All the datasets mentioned above are using BIO tagging scheme. Initially, we also included dataset BioNLP2004 [9] in our experiment. However, our resource for the experiment is not sufficient to handle the size of the dataset (22402 sentences). The train-test-validation split size table can be seen in table I.

TABLE I
Split Size Dataset Description

Dataset	Sentences Size		
	Train	*Test*	*Validation*
BTC	1,014	303	150
Tweebank	1,639	710	1,201

C. Experiment Settings

We held the experiment in Google Colab environment with one K80 GPU. We start with the default configuration of the pre-trained language model RoBERTa [10], then train it with given configuration . During training, we search for the best learning rate hyperparameter using GridSearch in range 10^{-4} to 10^{-3}. The complete hyperparameter settings can be seen at Table II.

TABLE II
Hyperparameter Settings Table

Hyperparameter	**Value**
Model	RoBERTa$_{LARGE}$
Epoch	10
Batch Size	32
Gradient Accumulation Steps	2
Weight Decay	None
Random Seed	42
N Max Config	1
CRF	True
Learning Rate	[10^{-4}, 10^{-3}]
Learning Rate Warmup Step Ratio	0.1

D. Results

The results of our experiment on BTC and Tweebank dataset can be seen in Table III and Table IV.

TABLE III
Result Comparison on BTC Dataset

Entity	**Metrics**			**Support**
	Precision	***Recall***	***F1-Score***	
Location	0.86	0.68	0.76	111
Organization	0.60	0.71	0.65	182
Person	**0.92**	**0.75**	**0.82**	**273**
Other	0.49	0.46	0.47	184
Micro Average	0.71	0.66	0.68	750
Macro Average	0.72	0.65	0.68	750

TABLE IV
Result Comparison on Tweebank Dataset

Entity	**Metrics**			**Support**
	Precision	***Recall***	***F1-Score***	
Location	0.74	0.74	0.74	636
Organization	0.72	0.64	0.68	1090
Person	**0.91**	**0.91**	**0.91**	**2650**
Micro Average	0.84	0.82	0.83	4376
Macro Average	0.79	0.76	0.77	4376

It can be seen from the table that the RoBERTa model recognize Person entity the best among other entities in all metrics. This is because that Person entity has the most support in both datasets. This implies the model has more chance to train with Person entity than other entities.

The gap between the micro average and the macro average of the metrics is wider in the Tweebank dataset compared to the BTC dataset. This is because the imbalance of the classes in the Tweebank dataset is more severe than the imbalance in the BTC dataset which increase the gap as micro-average scoring is sensitive to imbalance data.

E. Discussion

We can clearly see that the metrics score for the Tweebank dataset are overall better compared to the BTC dataset. The reason for this result difference is the Tweebank dataset has around 600 more sentences than BTC dataset. This means the Tweebank dataset has much more tokens that are likely be tagged as an entity compared to BTC dataset. Thus, the model is trained using more data when using Tweebank dataset compared to BTC dataset.

The most apparent weakness of the model is inability to be robust against class imbalance. We can infer this by looking at the gap between micro average and the macro average of the metrics as mentioned in the Result section.

V. Conclusions and Future Works

Overall, the results of this study demonstrate the effectiveness of the proposed NER framework in assisting pre-trained model finetuning process for a NER task. This is concluded from the comparison of our experiment using T-NER with the experiment conducted by the creators of T-NER. Despite a huge different in resources used in our experiment compared to the resources used in the original experiment, the performance in our experiment is only slightly worse than the original experiment's.

This experiment has shown that the proposed Python framework is an effective and efficient tool for fine-tuning BERT models for the NER task. Its ability to improve the performance of a model using its grid-search capability while reducing the amount of code required makes it a valuable resource for NLP researchers and practitioners. Future work should focus on extending the framework to support other NLP tasks and models, as well as exploring additional methods for improving model performance.

References

[1] A. Ushio and J. Camacho-Collados, "T-NER: An all-round python library for transformer-based named entity recognition," in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. Online: Association for Computational Linguistics, Apr. 2021, pp. 53–62. [Online]. Available: https://www.aclweb.org/anthology/2021.eacl-demos.7

[2] X. Carreras, L. Màrquez, and L. Padró, "A simple named entity extractor using adaboost," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, ser. CONLL '03. USA: Association for Computational Linguistics, 2003, p. 152–155. [Online]. Available: https://doi.org/10.3115/1119176.1119197

[3] Y. Sari, M. F. Hassan, and N. Zamin, "Rule-based pattern extractor and named entity recognition: A hybrid approach," in *2010 International Symposium on Information Technology*, vol. 2, 2010, pp. 563–568.

[4] D. Jurafsky and J. H. Martin, *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J.: Pearson Prentice Hall, 2009. [Online]. Available: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y

[5] F. Souza, R. Nogueira, and R. Lotufo, "Portuguese named entity recognition using bert-crf," 2019. [Online]. Available: https://arxiv.org/abs/1909.10649

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[7] L. Derczynski, K. Bontcheva, and I. Roberts, "Broad Twitter corpus: A diverse named entity recognition resource," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 1169–1179. [Online]. Available: https://aclanthology.org/C16-1111

[8] H. Jiang, Y. Hua, D. Beeferman, and D. Roy, "Annotating the tweebank corpus on named entity recognition and building NLP models for social media analysis," *CoRR*, vol. abs/2201.07281, 2022. [Online]. Available: https://arxiv.org/abs/2201.07281

[9] N. Collier and J.-D. Kim, "Introduction to the bio-entity recognition task at JNLPBA," in *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP)*. Geneva, Switzerland: COLING, Aug. 28th and 29th 2004, pp. 73–78. [Online]. Available: https://aclanthology.org/W04-1213

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available: http://arxiv.org/abs/1907.11692