HW 4 - POS Tagging with Hugging Face

In this exercise, you will create a part-of-speech (POS) tagging system for Thai text using NECTEC's ORCHID corpus. Instead of building your own deep learning architecture from scratch, you will leverage a pretrained tokenizer and a pretrained token classification model from Hugging Face.

We have provided some starter code for data cleaning and preprocessing in this notebook, but feel free to modify those parts to suit your needs. You are welcome to use additional libraries (e.g., scikit-learn) as long as you incorporate the pretrained Hugging Face model. Specifically, you will need to:

1. Load a pretrained tokenizer and token classification model.
2. Fine-tune it on the ORCHID corpus for POS tagging.
3. Evaluate and report the performance of your model on the test data.

Don't forget to change hardware accelrator to GPU in runtime on Google Colab

1. Setup and Preprocessing

```
# Install transformers and thai2transformers
!pip install -q wandb
!pip install -q transformers==4.30.1 datasets evaluate thaixtransformers
!pip install -q emoji pythainlp sefr_cut tinydb seqeval sentencepiece pydantic jsonlines
!pip install -q peft==0.10.0
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 113.6/113.6 kB 634.7 kB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.2/7.2 MB 46.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 480.6/480.6 kB 25.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.0/84.0 kB 4.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 5.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 179.3/179.3 kB 12.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 9.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.9/17.9 MB 39.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 48.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 590.6/590.6 kB 19.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 16.3 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviou
sentence-transformers 3.3.1 requires transformers<5.0.0,>=4.41.0, but you have transformers 4.30.1 which is incompatible
torch 2.5.1+cu124 requires nvidia-cublas-cu12==12.4.5.8; platform_system == "Linux" and platform_machine == "x86_64", bu
torch 2.5.1+cu124 requires nvidia-cuda-cupti-cu12==12.4.127; platform_system == "Linux" and platform_machine == "x86_64"
torch 2.5.1+cu124 requires nvidia-cuda-nvrtc-cu12==12.4.127; platform_system == "Linux" and platform_machine == "x86_64"
torch 2.5.1+cu124 requires nvidia-cuda-runtime-cu12==12.4.127; platform_system == "Linux" and platform_machine == "x86_6
torch 2.5.1+cu124 requires nvidia-cudnn-cu12==9.1.0.70; platform_system == "Linux" and platform_machine == "x86_64", but
torch 2.5.1+cu124 requires nvidia-cufft-cu12==11.2.1.3; platform_system == "Linux" and platform_machine == "x86_64", but
torch 2.5.1+cu124 requires nvidia-curand-cu12==10.3.5.147; platform_system == "Linux" and platform_machine == "x86_64",
torch 2.5.1+cu124 requires nvidia-cusolver-cu12==11.6.1.9; platform_system == "Linux" and platform_machine == "x86_64",
torch 2.5.1+cu124 requires nvidia-cusparse-cu12==12.3.1.170; platform_system == "Linux" and platform_machine == "x86_64"
torch 2.5.1+cu124 requires nvidia-nvjitlink-cu12==12.4.127; platform_system == "Linux" and platform_machine == "x86_64",
gcsfs 2024.10.0 requires fsspec==2024.10.0, but you have fsspec 2024.9.0 which is incompatible.
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.6/43.6 kB 1.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 54.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 118.3/118.3 kB 11.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 70.4 MB/s eta 0:00:00
  Building wheel for seqeval (setup.py) ... done
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.1/199.1 kB 5.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 64.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 35.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 49.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 13.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 3.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 81.0 MB/s eta 0:00:00
```

Setup

1. Register Wandb account (and confirm your email)
2. `wandb login` and copy paste the API key when prompt

```
import wandb
```

```
wandb.login(key=r'648f0ebca50c7021eefe306ab62fcbf0029574da')
```

```
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: jirayuwat12 (myfistteam) to https://api.wandb.ai. Use `wandb login --relogin` to force re
True
```

We encourage you to login to your `Hugging Face` account so you can upload and share your model with the community. When prompted, enter your token to login

```
%pip install -q ipywidgets
from huggingface_hub import notebook_login

notebook_login()
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 18.0 MB/s eta 0:00:00
```

Download the dataset from Hugging Face

```
from datasets import load_dataset

orchid = load_dataset("Thichow/orchid_corpus")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens),
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md: 100%                                                  79.0/79.0 [00:00<00:00, 5.69kB/s]
orchid_corpus.py: 100%                                           7.91k/7.91k [00:00<00:00, 525kB/s]
The repository for Thichow/orchid_corpus contains custom code which must be executed to correctly load the dataset. You
You can avoid this prompt in future by passing the argument `trust_remote_code=True`.

Do you wish to run the custom code? [y/N] y
Downloading data: 100%                                           5.24M/5.24M [00:00<00:00, 69.6MB/s]
Downloading data: 100%                                           1.36M/1.36M [00:00<00:00, 62.9MB/s]
Generating train split:        18500/0 [00:01<00:00, 12576.35 examples/s]
Generating test split:         4625/0 [00:00<00:00, 10888.97 examples/s]
```

```
orchid
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence'],
        num_rows: 18500
    })
    test: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence'],
        num_rows: 4625
    })
})
```

```
orchid['train'][0]
```

```
{'id': '0',
 'label_tokens': ['การ', 'ประชุม', 'ทาง', 'วิชาการ', ' ', 'ครั้ง', 'ที่ 1'],
 'pos_tags': [21, 39, 26, 26, 37, 4, 18],
 'sentence': 'การประชุมทางวิชาการ ครั้งที่ 1'}
```

```
orchid['train'][0]["sentence"]
```

```
'การประชุมทางวิชาการ ครั้งที่ 1'
```

```
''.join(orchid['train'][0]['label_tokens'])
```

```
'การประชุมทางวิชาการ ครั้งที่ 1'
```

```
label_list = orchid["train"].features[f"pos_tags"].feature.names
print('total type of pos_tags :', len(label_list))
print(label_list)
```

```
total type of pos_tags : 47
['ADVI', 'ADVN', 'ADVP', 'ADVS', 'CFQC', 'CLTV', 'CMTR', 'CMTR@PUNC', 'CNIT', 'CVBL', 'DCNM', 'DDAC', 'DDAN', 'DDAQ', 'D
```

```
import numpy as np
import numpy.random
import torch

from tqdm.auto import tqdm
from functools import partial

#transformers
from transformers import (
    CamembertTokenizer,
    AutoTokenizer,
    AutoModel,
    AutoModelForMaskedLM,
    AutoModelForSequenceClassification,
    AutoModelForTokenClassification,
    TrainingArguments,
    Trainer,
    pipeline,
)

#thaixtransformers
from thaixtransformers import Tokenizer
from thaixtransformers.preprocess import process_transformers
```

```
The cache for model files in Transformers v4.22.0 has been updated. Migrating your old cache. This is a one-time only op
0/0 [00:00<?, ?it/s]
```

Next, we load a pretrained tokenizer from Hugging Face. In this work, we utilize WangchanBERTa, a Thai-specific pretrained model, as the tokenizer.

Choose Pretrained Model

In this notebook, you can choose from 5 versions of WangchanBERTa, XLMR and mBERT to perform downstream tasks on Thai datasets. The datasets are:

- `wangchanberta-base-att-spm-uncased` (recommended) - Largest WangchanBERTa trained on 78.5GB of Assorted Thai Texts with subword tokenizer SentencePiece
- `xlm-roberta-base` - Facebook's XLMR trained on 100 languages
- `bert-base-multilingual-cased` - Google's mBERT trained on 104 languages
- `wangchanberta-base-wiki-newmm` - WangchanBERTa trained on Thai Wikipedia Dump with PyThaiNLP's word-level tokenizer `newmm`
- `wangchanberta-base-wiki-syllable` - WangchanBERTa trained on Thai Wikipedia Dump with PyThaiNLP's syllabel-level tokenizer `syllable`
- `wangchanberta-base-wiki-sefr` - WangchanBERTa trained on Thai Wikipedia Dump with word-level tokenizer `SEFR`
- `wangchanberta-base-wiki-spm` - WangchanBERTa trained on Thai Wikipedia Dump with subword-level tokenizer SentencePiece

In the first part, we require you to select the wangchanberta-base-att-spm-uncased.

Learn more about using wangchanberta at wangchanberta_getting_started_ai_reseach

- **You need to set the transformers version to transformers==4.30.1.**

```
In the first part, we require you to select the wangchanberta-base-att-spm-uncased.
```

Choose Pretrained Model

```
model_names = [
    'airesearch/wangchanberta-base-att-spm-uncased',
    'airesearch/wangchanberta-base-wiki-newmm',
    'airesearch/wangchanberta-base-wiki-ssg',
    'airesearch/wangchanberta-base-wiki-sefr',
    'airesearch/wangchanberta-base-wiki-spm',
]

#@title Choose Pretrained Model
model_name = "airesearch/wangchanberta-base-att-spm-uncased"

#create tokenizer
tokenizer = Tokenizer(model_name).from_pretrained(
                f'{model_name}',
```

```
                revision='main',
                model_max_length=416,)

```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.py:795: FutureWarning: `resume_download` is deprec
  warnings.warn(
```

```
sentencepiece.bpe.model: 100%  905k/905k [00:00<00:00, 4.76MB/s]

tokenizer_config.json: 100%  282/282 [00:00<00:00, 10.2kB/s]

config.json: 100%  546/546 [00:00<00:00, 34.6kB/s]
```

```
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may
The tokenizer class you load from this checkpoint is 'CamembertTokenizer'.
The class this function is called from is 'WangchanbertaTokenizer'.
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may
The tokenizer class you load from this checkpoint is 'CamembertTokenizer'.
The class this function is called from is 'WangchanbertaTokenizer'.
```

Let's try using a pretrained tokenizer.

```
text = 'ศิลปะไม่เป็นเจ้านายใคร และไม่เป็นขี้ข้าใคร'
print('text :', text)
tokens = []
for i in tokenizer([text], is_split_into_words=True)['input_ids']:
  tokens.append(tokenizer.decode(i))
print('tokens :', tokens)
```

```
text : ศิลปะไม่เป็นเจ้านายใคร และไม่เป็นขี้ข้าใคร
tokens : ['<s>', '', 'ศิลปะ', 'ไม่เป็น', 'เจ้านาย', 'ใคร', '<_>', 'และ', 'ไม่เป็น', 'ขี้ข้า', 'ใคร', '</s>']
```

model : * `wangchanberta-base-att-spm-uncased`

First, we print examples of label tokens from our dataset for inspection.

```
example = orchid["train"][0]
for i in example :
    print(i, ':', example[i])
```

```
id : 0
label_tokens : ['การ', 'ประชุม', 'ทาง', 'วิชาการ', ' ', 'ครั้ง', 'ที่ 1']
pos_tags : [21, 39, 26, 26, 37, 4, 18]
sentence : การประชุมทางวิชาการ ครั้งที่ 1
```

Then, we use the sentence '**การประชุมทางวิชาการครั้งที่** 1' to be tokenized by the pretrained tokenizer model.

```
text = 'การประชุมทางวิชาการ ครั้งที่ 1'
tokenizer(text)
```

```
{'input_ids': [5, 10, 882, 8222, 8, 10, 1014, 8, 10, 59, 6], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

These are already mapped into discrete values. We can uncover the original token text from the tokens by.

```
for i in tokenizer(text)['input_ids']:
  print(tokenizer.convert_ids_to_tokens(i))
```

```
<s>
▁
การประชุม
ทางวิชาการ
<_>
▁
ครั้งที่
<_>
▁
1
</s>
```

Now let's look at another example.

```
example = orchid["train"][1899]
print('sentence :', example["sentence"])
tokenized_input = tokenizer([example["sentence"]], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
print('tokens :',tokens)
print('label tokens :', example["label_tokens"])
print('label pos :', example["pos_tags"])
```

```
sentence : โดยพิจารณาจากพจนานุกรมภาษาคู่ (Bilingual transfer dictionary)
tokens : ['<s>', '_โดย', 'พิจารณาจาก', 'พจนานุกรม', 'ภาษา', 'คู่', '<_>', '_(', '<unk>', 'i', 'ling', 'ual', '<_>', '_', 'tra
label tokens : ['โดย', 'พิจารณา', 'จาก', 'พจนานุกรม', 'ภาษา', 'คู่', ' ', '(', 'Bilingual transfer dictionary', ')']
label pos : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
```

Notice how `B` becomes an `<unk>` token. This is because this is an uncased model, meaning it only handles small English characters.

#TODO 0

Convert the dataset to lowercase.

```
# Create a lowercase dataset for uncased BERT
def lower_case_sentences(examples):
  lower_cased_examples = examples
  lower_cased_examples['sentence'] = lower_cased_examples['sentence'].lower()
  lower_cased_examples['label_tokens'] = [t.lower() for t in lower_cased_examples['label_tokens']]

  # fill code here to lower case the "sentence" and "label_tokens"

  return lower_cased_examples
```

```
orchidl = orchid.map(lower_case_sentences)
```

```
Map: 100%  18500/18500 [00:04<00:00, 6222.74 examples/s]
Map: 100%  4625/4625 [00:00<00:00, 11289.08 examples/s]
```

```
orchidl
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence'],
        num_rows: 18500
    })
    test: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence'],
        num_rows: 4625
    })
})
```

```
orchidl["train"][1899]
```

```
{'id': '1899',
 'label_tokens': ['โดย',
  'พิจารณา',
  'จาก',
  'พจนานุกรม',
  'ภาษา',
  'คู่',
  ' ',
  '(',
  'bilingual transfer dictionary',
  ')'],
 'pos_tags': [25, 39, 38, 26, 26, 5, 37, 37, 26, 37],
 'sentence': 'โดยพิจารณาจากพจนานุกรมภาษาคู่ (bilingual transfer dictionary)'}
```

Now let's examine the labels again.

```
example = orchidl["train"][1899]
print('sentence :', example["sentence"])
tokenized_input = tokenizer([example["sentence"]], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
print('tokens :',tokens)
print('label tokens :', example["label_tokens"])
print('label pos :', example["pos_tags"])
```

```
sentence : โดยพิจารณาจากพจนานุกรมภาษาคู่ (bilingual transfer dictionary)
tokens : ['<s>', '_โดย', 'พิจารณาจาก', 'พจนานุกรม', 'ภาษา', 'คู่', '<_>', '_(', 'bi', 'ling', 'ual', '<_>', '_', 'trans', 'f
label tokens : ['โดย', 'พิจารณา', 'จาก', 'พจนานุกรม', 'ภาษา', 'คู่', ' ', '(', 'bilingual transfer dictionary', ')']
label pos : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
```

```
example = orchidl["train"][0]
print('sentence :', example["sentence"])
tokenized_input = tokenizer([example["sentence"]], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
print('tokens :',tokens)
```

```
print('label tokens :', example["label_tokens"])
print('label pos :', example["pos_tags"])
```

```
sentence : การประชุมทางวิชาการ ครั้งที่ 1
tokens : ['<s>', '_', 'การประชุม', 'ทางวิชาการ', '<_>', '_', 'ครั้งที่', '<_>', '_', '1', '</s>']
label tokens : ['การ', 'ประชุม', 'ทาง', 'วิชาการ', ' ', 'ครั้ง', 'ที่ 1']
label pos : [21, 39, 26, 26, 37, 4, 18]
```

In the example above, tokens refer to those tokenized using the pretrained tokenizer, while label tokens refer to tokens tokenized from our dataset.

Do you see something?

Yes, the tokens from the two tokenizers do not match.

- sentence : `การประชุมทางวิชาการ ครั้งที่ 1`

- tokens : `['<s>', '_', 'การประชุม', 'ทางวิชาการ', '<_>', '_', 'ครั้งที่', '<_>', '_', '1', '</s>']`

- label tokens : `['การ', 'ประชุม', 'ทาง', 'วิชาการ', ' ', 'ครั้ง', 'ที่ 1']`
- label pos : `[21, 39, 26, 26, 37, 4, 18]`

You can see that in our label tokens, 'การ' has a POS tag of 21, and 'ประชุม' has a POS tag of 39. However, when we tokenize the sentence using WangchanBERTa, we get the token 'การประชุม'. What POS tag should we assign to this new token?

What should we do ?

Based on this example, we found that the tokens from the WangchanBERTa do not directly align with our label tokens. This means we cannot directly use the label POS tags. Therefore, we need to reassign POS tags to the tokens produced by WangchanBERTa tokenization. The method we will use is majority voting:

- If a token from the WangchanBERTa matches a label token exactly, we will directly assign the POS tag from the label POS.
- If the token generated overlaps or combines multiple label tokens, we assign the POS tag based on the number of characters in each token: If the token contains the most characters from any label token, we assign the POS tag from that label token.

Example :

```
# "การประชุม" (9 chars) is formed from "การ" (3 chars) + "ประชุม" (6 chars).
# "การ" has a POS tag of 21,
# and "ประชุม" has a POS tag of 39.
# Therefore, the POS tag for "การประชุม" is 39,
# as "การประชุม" is derived more from the "ประชุม" part than from the "การ" part.

# 'ทางวิชาการ' (10 chars) is formed from 'ทาง' (3 chars) + 'วิชาการ' (7 chars)
# "ทาง" has a POS tag of 26,
# and "วิชาการ" has a POS tag of 2.
# Therefore, the POS tag for "ทางวิชาการ" is 2,
# as "ทางวิชาการ" is derived more from the "ทาง" part than from the "วิชาการ" part.
```

#TODO 1

Warning: Please be careful of <unk>, an unknown word token.

Warning: Please be careful of " ำ ", the 'am' vowel. WangchanBERTa's internal preprocessing replaces all " ำ " to ' ' and 'า'

Assigning the label -100 to the special tokens `[<s>]` and `[</s>]` and `[_]` so they're ignored by the PyTorch loss function (see CrossEntropyLoss: ignore_index)

```
def majority_vote_pos(examples):

    ##########################################################################################################################
    # TO DO: Since the tokens from the output of the pretrained tokenizer
    # do not match the tokens in the label tokens of the dataset,
    # the task is to create a function to determine the POS tags of the tokens generated by the pretrained tokenizer.
    # This should be done by referencing the POS tags in the label tokens. If a token partially overlaps with others,
    # the POS tag from the segment with the greater number of characters should be assigned.
    #
    # Example :
    # "การประชุม" (9 chars) is formed from "การ" (3 chars) + "ประชุม" (6 chars).
    # "การ" has a POS tag of 21,
    # and "ประชุม" has a POS tag of 39.
```

```
    # Therefore, the POS tag for "การประชุม" is 39,
    # as "การประชุม" is derived more from the "ประชุม" part than from the "การ" part.
    #
    # 'ทางวิชาการ' (10 chars) is formed from 'ทาง' (3 chars) + 'วิชาการ' (7 chars)
    # "ทาง" has a POS tag of 26,
    # and "วิชาการ" has a POS tag of 2.
    # Therefore, the POS tag for "ทางวิชาการ" is 2,
    # as "ทางวิชาการ" is derived more from the "ทาง" part than from the "วิชาการ" part.

    # tokenize word by pretrained tokenizer
    tokenized_inputs = tokenizer([examples["sentence"]], is_split_into_words=True)

    # FILL CODE HERE
    label_tokens = examples["label_tokens"]
    pos_tags = examples["pos_tags"]
    new_pos_result = []

    new_tokens = tokenizer.convert_ids_to_tokens(tokenized_inputs["input_ids"])
    # print(new_tokens, "\n", label_tokens, "\n", pos_tags)

    label_idx = 0
    i = 0
    for t in new_tokens:
        if t in ["<s>", "</s>", "_"]:
            new_pos_result.append(-100)
            continue
        # if t == "<_>":
        #     new_pos_result.append(37)
        #     continue

        buffer = ""
        weights = {}
        t = t.replace('ๆ', "ๆ")
        t = t.replace("<_>", " ")
        if t[0] == "_":
            t = t[1:]
        while label_tokens[label_idx][i] != t[0]:
            i += 1
            if i == len(label_tokens[label_idx]):
                label_idx += 1
                i = 0

        while buffer != t:
            buffer += label_tokens[label_idx][i]
            # print(t, buffer, t==buffer)
            if pos_tags[label_idx] not in weights:
                weights[pos_tags[label_idx]] = 0
            weights[pos_tags[label_idx]] += 1
            i += 1
            if i == len(label_tokens[label_idx]):
                label_idx += 1
                i = 0
        # print(weights)
        max_key = max(weights, key=weights.get)
        new_pos_result.append(max_key)

    tokenized_inputs['tokens'] = new_tokens
    tokenized_inputs['labels'] = new_pos_result

    return tokenized_inputs
    ##########################################################################################################################
```

```
tokenized_orchid = orchidl.map(majority_vote_pos)
```

```
Map: 100%                 18500/18500 [00:22<00:00, 2113.48 examples/s]
Map: 100%                 4625/4625 [00:02<00:00, 2126.82 examples/s]
```

```
tokenized_orchid
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence', 'input_ids', 'attention_mask', 'tokens', 'labels'],
        num_rows: 18500
    })
    test: Dataset({
        features: ['id', 'label_tokens', 'pos_tags', 'sentence', 'input_ids', 'attention_mask', 'tokens', 'labels'],
        num_rows: 4625
    })
})
```

```
1 tokenized_orchid['train'][0]
```

```
{'id': '0',
 'label_tokens': ['การ', 'ประชุม', 'ทาง', 'วิชาการ', ' ', 'ครั้ง', 'ที่ 1'],
 'pos_tags': [21, 39, 26, 26, 37, 4, 18],
 'sentence': 'การประชุมทางวิชาการ ครั้งที่ 1',
 'input_ids': [5, 10, 882, 8222, 8, 10, 1014, 8, 10, 59, 6],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 'tokens': ['<s>',
  '_',
  'การประชุม',
  'ทางวิชาการ',
  '<_>',
  '_',
  'ครั้งที่',
  '<_>',
  '_',
  '1',
  '</s>'],
 'labels': [-100, -100, 39, 26, 37, -100, 4, 18, -100, 18, -100]}
```

```
1 example = tokenized_orchid["train"][0]
2 for i in example :
3     print(i, ":", example[i])
```

```
id : 0
label_tokens : ['การ', 'ประชุม', 'ทาง', 'วิชาการ', ' ', 'ครั้ง', 'ที่ 1']
pos_tags : [21, 39, 26, 26, 37, 4, 18]
sentence : การประชุมทางวิชาการ ครั้งที่ 1
input_ids : [5, 10, 882, 8222, 8, 10, 1014, 8, 10, 59, 6]
attention_mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
tokens : ['<s>', '_', 'การประชุม', 'ทางวิชาการ', '<_>', '_', 'ครั้งที่', '<_>', '_', '1', '</s>']
labels : [-100, -100, 39, 26, 37, -100, 4, 18, -100, 18, -100]
```

This is the result after we realigned the POS based on the majority vote.

- label_tokens : `['การ', 'ประชุม', 'ทาง', 'วิชาการ', ' ', 'ครั้ง', 'ที่ 1']`
- pos_tags : `[21, 39, 26, 26, 37, 4, 18]`
- tokens : `['<s>', '_', 'การประชุม', 'ทางวิชาการ', '<_>', '_', 'ครั้งที่', '<_>', '_', '1', '</s>']`
- labels : `[-100, -100, 39, 26, 37, -100, 4, 18, -100, 18, -100]`

`['<s>', '_', '</s>'] : -100`

Check :

> "การประชุม" (9 chars) is formed from "การ" (3 chars) + "ประชุม" (6 chars).
>
> "การ" has a POS tag of 21,
>
> and "ประชุม" has a POS tag of 39.
>
> Therefore, the POS tag for "การประชุม" is 39,
>
> as "การประชุม" is derived more from the "ประชุม" part than from the "การ" part.

```
1 # hard test case
2 example = tokenized_orchid["train"][1899]
3 for i in example :
4     print(i, ":", example[i])
```

```
id : 1899
label_tokens : ['โดย', 'พิจารณา', 'จาก', 'พจนานุกรม', 'ภาษา', 'คู่', ' ', '(', 'bilingual transfer dictionary', ')']
pos_tags : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
sentence : โดยพิจารณาจากพจนานุกรมภาษาคู่ (bilingual transfer dictionary)
input_ids : [5, 489, 15617, 19737, 958, 493, 8, 1241, 4906, 11608, 12177, 8, 10, 11392, 9806, 8, 10, 2951, 15779, 8001,
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]
tokens : ['<s>', '_โดย', 'พิจารณาจาก', 'พจนานุกรม', 'ภาษา', 'คู่', '<_>', '_(', 'bi', 'ling', 'ual', '<_>', '_', 'trans', 'f
labels : [-100, 25, 39, 26, 26, 5, 37, 37, 26, 26, 26, 26, -100, 26, 26, 26, -100, 26, 26, 26, 37, -100]
```

Expected output

```
id : 1899
label_tokens : ['โดย', 'พิจารณา', 'จาก', 'พจนานุกรม', 'ภาษา', 'คู่', ' ', '(', 'bilingual transfer dictionary', ')']
pos_tags : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
sentence : โดยพิจารณาจากพจนานุกรมภาษาคู่ (bilingual transfer dictionary)
input_ids : [5, 489, 15617, 19737, 958, 493, 8, 1241, 4906, 11608, 12177, 8, 10, 11392, 9806, 8, 10, 2951, 15779, 8001, 29, 6]
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]
tokens : ['<s>', '_โดย', 'พิจารณาจาก', 'พจนานุกรม', 'ภาษา', 'คู่', '<_>', '_(', 'bi', 'ling', 'ual', '<_>', '_', 'trans', 'fer', '<_>',
labels : [-100, 25, 39, 26, 26, 5, 37, 37, 26, 26, 26, 26, -100, 26, 26, 26, -100, 26, 26, 26, 37, -100]
```

Train and Evaluate model

We will create a batch of examples using DataCollatorWithPadding.

Data collators are objects that will form a batch by using a list of dataset elements as input. These elements are of the same type as the elements of train_dataset or eval_dataset.

DataCollatorWithPadding will help us pad the sentences to the longest length in a batch during collation, instead of padding the whole dataset to the maximum length. This allows for efficient computation during each batch.

- DataCollatorForTokenClassification : `padding (bool, str or PaddingStrategy, optional, defaults to True)`
- `True` or `'longest'` (default): Pad to the longest sequence in the batch (or no padding if only a single sequence is provided).

```
from transformers import DataCollatorForTokenClassification

data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)
```

For evaluating your model's performance. You can quickly load a evaluation method with the Evaluate library. For this task, load the seqeval framework (see the Evaluate quick tour to learn more about how to load and compute a metric). Seqeval actually produces several scores: precision, recall, F1, and accuracy.

```
import evaluate

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

```
Downloading builder script: 100%  6.34k/6.34k [00:00<00:00, 295kB/s]
```

Huggingface requires us to write a `compute_metrics()` function. This will be invoked when huggingface evalutes a model.

Note that we ignore to evaluate on -100 labels.

```
import numpy as np
import warnings


def compute_metrics(p):
    predictions, labels = p
    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)
    ]

    with warnings.catch_warnings():
        warnings.filterwarnings("ignore")
        results = seqeval.compute(predictions=true_predictions, references=true_labels)
    return {
        "precision": results["overall_precision"],
        "recall": results["overall_recall"],
        "f1": results["overall_f1"],
        "accuracy": results["overall_accuracy"],
    }
```

The total number of labels in our POS tag set.

```
id2label = {
    0: 'ADVI',
    1: 'ADVN',
    2: 'ADVP',
    3: 'ADVS',
    4: 'CFQC',
    5: 'CLTV',
    6: 'CMTR',
    7: 'CMTR@PUNC',
    8: 'CNIT',
    9: 'CVBL',
    10: 'DCNM',
```

```
    11: 'DDAC',
    12: 'DDAN',
    13: 'DDAQ',
    14: 'DDBQ',
    15: 'DIAC',
    16: 'DIAQ',
    17: 'DIBQ',
    18: 'DONM',
    19: 'EAFF',
    20: 'EITT',
    21: 'FIXN',
    22: 'FIXV',
    23: 'JCMP',
    24: 'JCRG',
    25: 'JSBR',
    26: 'NCMN',
    27: 'NCNM',
    28: 'NEG',
    29: 'NLBL',
    30: 'NONM',
    31: 'NPRP',
    32: 'NTTL',
    33: 'PDMN',
    34: 'PNTR',
    35: 'PPRS',
    36: 'PREL',
    37: 'PUNC',
    38: 'RPRE',
    39: 'VACT',
    40: 'VATT',
    41: 'VSTA',
    42: 'XVAE',
    43: 'XVAM',
    44: 'XVBB',
    45: 'XVBM',
    46: 'XVMM',
    # 47: 'O'
}
label2id = {}
for k, v in id2label.items() :
    label2id[v] = k

label2id
```

```
{'ADVI': 0,
 'ADVN': 1,
 'ADVP': 2,
 'ADVS': 3,
 'CFQC': 4,
 'CLTV': 5,
 'CMTR': 6,
 'CMTR@PUNC': 7,
 'CNIT': 8,
 'CVBL': 9,
 'DCNM': 10,
 'DDAC': 11,
 'DDAN': 12,
 'DDAQ': 13,
 'DDBQ': 14,
 'DIAC': 15,
 'DIAQ': 16,
 'DIBQ': 17,
 'DONM': 18,
 'EAFF': 19,
 'EITT': 20,
 'FIXN': 21,
 'FIXV': 22,
 'JCMP': 23,
 'JCRG': 24,
 'JSBR': 25,
 'NCMN': 26,
 'NCNM': 27,
 'NEG': 28,
 'NLBL': 29,
 'NONM': 30,
 'NPRP': 31,
 'NTTL': 32,
 'PDMN': 33,
 'PNTR': 34,
 'PPRS': 35,
 'PREL': 36,
 'PUNC': 37,
 'RPRE': 38,
 'VACT': 39,
 'VATT': 40,
 'VSTA': 41,
```

```
    'XVAE': 42,
    'XVAM': 43,
    'XVBB': 44,
    'XVBM': 45,
    'XVMM': 46}
```

```
labels = [i for i in id2label.values()]
labels
```

```
['ADVI',
 'ADVN',
 'ADVP',
 'ADVS',
 'CFQC',
 'CLTV',
 'CMTR',
 'CMTR@PUNC',
 'CNIT',
 'CVBL',
 'DCNM',
 'DDAC',
 'DDAN',
 'DDAQ',
 'DDBQ',
 'DIAC',
 'DIAQ',
 'DIBQ',
 'DONM',
 'EAFF',
 'EITT',
 'FIXN',
 'FIXV',
 'JCMP',
 'JCRG',
 'JSBR',
 'NCMN',
 'NCNM',
 'NEG',
 'NLBL',
 'NONM',
 'NPRP',
 'NTTL',
 'PDMN',
 'PNTR',
 'PPRS',
 'PREL',
 'PUNC',
 'RPRE',
 'VACT',
 'VATT',
 'VSTA',
 'XVAE',
 'XVAM',
 'XVBB',
 'XVBM',
 'XVMM']
```

Load pretrained model

Select a pretrained model for fine-tuning to develop a POS Tagger model using the Orchid corpus dataset.

- model : `wangchanberta-base-att-spm-uncased`
- Don't forget to update the num_labels.

You're ready to start training your model now! Load pretrained model with AutoModelForTokenClassification along with the number of expected labels, and the label mappings:

`In the first part, we require you to select the wangchanberta-base-att-spm-uncased.`

Choose Pretrained Model

```
model_names = [
    'wangchanberta-base-att-spm-uncased',
    'wangchanberta-base-wiki-newmm',
    'wangchanberta-base-wiki-ssg',
    'wangchanberta-base-wiki-sefr',
    'wangchanberta-base-wiki-spm',
]

#@title Choose Pretrained Model
model_name = "wangchanberta-base-att-spm-uncased"
```

```
#create model
model = AutoModelForTokenClassification.from_pretrained(
    f"airesearch/{model_name}",
    revision='main',
    num_labels=47, id2label=id2label, label2id=label2id
)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.py:795: FutureWarning: `resume_download` is deprec
  warnings.warn(
model.safetensors: 100%                                423M/423M [00:02<00:00, 231MB/s]
Some weights of the model checkpoint at airesearch/wangchanberta-base-att-spm-uncased were not used when initializing Ca
- This IS expected if you are initializing CamembertForTokenClassification from the checkpoint of a model trained on ano
- This IS NOT expected if you are initializing CamembertForTokenClassification from the checkpoint of a model that you e
Some weights of CamembertForTokenClassification were not initialized from the model checkpoint at airesearch/wangchanber
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

#TODO 2

- Configure your training hyperparameters using `**TrainingArguments**`. The only required parameter is is `output_dir`, which determines the directory where your model will be saved. To upload the model to the Hugging Face Hub, set push_to_hub=True (note: you must be logged into Hugging Face for this). During training, the Trainer will compute seqeval metrics at the end of each epoch and store the training checkpoint.
- Provide the `**Trainer**` with the training arguments, as well as the model, dataset, tokenizer, data collator, and compute_metrics function.
- Use `**train()**` to fine-tune the model.

Read huggingface's tutorial for more details.

```
training_args = TrainingArguments(
    ########################
    output_dir="pos-spm-uncased",
    learning_rate=2e-5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    num_train_epochs=2,
    weight_decay=0.01,
    push_to_hub=True
    ########################
)

trainer = Trainer(
    ########################
    model=model,
    args=training_args,
    train_dataset=tokenized_orchid["train"],
    eval_dataset=tokenized_orchid["test"],
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    ########################
)

trainer.train()
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_deprecation.py:131: FutureWarning: 'Repository' (from 'hu
For more details, please read https://huggingface.co/docs/huggingface_hub/concepts/git_vs_http.
  warnings.warn(warning_message, FutureWarning)
Cloning https://huggingface.co/Jirayuwat12/pos-spm-uncased into local empty directory.
WARNING:huggingface_hub.repository:Cloning https://huggingface.co/Jirayuwat12/pos-spm-uncased into local empty directory
/usr/local/lib/python3.11/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation of AdamW is
  warnings.warn(
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
Tracking run with wandb version 0.19.5
Run data is saved locally in /content/wandb/run-20250201_083555-7n089k1e
Syncing run breezy-vortex-16 to Weights & Biases (docs)
View project at https://wandb.ai/myfistteam/huggingface
View run at https://wandb.ai/myfistteam/huggingface/runs/7n089k1e
[1158/1158 07:25, Epoch 2/2]
```

Step	Training Loss
500	0.984700
1000	0.361100

```
TrainOutput(global_step=1158, training_loss=0.6244609681229929, metrics={'train_runtime': 450.5004,
'train_samples_per_second': 82.131, 'train_steps_per_second': 2.57, 'total_flos': 1207602506181672.0, 'train_loss':
0.6244609681229929, 'epoch': 2.0})
```

Inference

With your model fine-tuned, you can now perform inference.

```
text = "การประชุมทางวิชาการ ครั้งที่ 1"
```

In the first part, we require you to select the wangchanberta-base-att-spm-uncased.

Choose Pretrained Model

```
from transformers import AutoTokenizer

# Load pretrained tokenizer from Hugging Face
#@title Choose Pretrained Model
model_name = "airesearch/wangchanberta-base-att-spm-uncased"

tokenizer = Tokenizer(model_name).from_pretrained(model_name)
inputs = tokenizer(text, return_tensors="pt")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.py:795: FutureWarning: `resume_download` is deprec
  warnings.warn(
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may
The tokenizer class you load from this checkpoint is 'CamembertTokenizer'.
The class this function is called from is 'WangchanbertaTokenizer'.
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may
The tokenizer class you load from this checkpoint is 'CamembertTokenizer'.
The class this function is called from is 'WangchanbertaTokenizer'.
```

```
inputs
```

```
{'input_ids': tensor([[   5,   10,  882, 8222,    8,   10, 1014,    8,   10,   59,    6]]), 'attention_mask':
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

```
from transformers import AutoModelForTokenClassification

## Load your fine-tuned model from Hugging Face
model = AutoModelForTokenClassification.from_pretrained("jirayuwat12/pos-spm-uncased") ## your model path from Hugging Fa
with torch.no_grad():
    logits = model(**inputs).logits
```

```
config.json: 100%                                   2.41k/2.41k [00:00<00:00, 71.1kB/s]
pytorch_model.bin: 100%                             419M/419M [00:09<00:00, 42.7MB/s]
/usr/local/lib/python3.11/dist-packages/transformers/modeling_utils.py:463: FutureWarning: You are using `torch.load` wi
  return torch.load(checkpoint_file, map_location="cpu")
```

```
predictions = torch.argmax(logits, dim=2)
predicted_token_class = [model.config.id2label[t.item()] for t in predictions[0]]
predicted_token_class
```

```
['PUNC',
 'PUNC',
 'VACT',
 'NCMN',
 'PUNC',
 'PUNC',
 'CFQC',
 'DONM',
 'DONM',
 'DONM',
 'PUNC']
```

```
id2label
```

```
{0: 'ADVI',
 1: 'ADVN',
 2: 'ADVP',
 3: 'ADVS',
 4: 'CFQC',
 5: 'CLTV',
 6: 'CMTR',
 7: 'CMTR@PUNC',
 8: 'CNIT',
 9: 'CVBL',
 10: 'DCNM',
 11: 'DDAC',
 12: 'DDAN',
 13: 'DDAQ',
 14: 'DDBQ',
 15: 'DIAC',
 16: 'DIAQ',
 17: 'DIBQ',
 18: 'DONM',
 19: 'EAFF',
 20: 'EITT',
 21: 'FIXN',
 22: 'FIXV',
 23: 'JCMP',
 24: 'JCRG',
 25: 'JSBR',
 26: 'NCMN',
 27: 'NCNM',
 28: 'NEG',
 29: 'NLBL',
 30: 'NONM',
 31: 'NPRP',
 32: 'NTTL',
 33: 'PDMN',
 34: 'PNTR',
 35: 'PPRS',
 36: 'PREL',
 37: 'PUNC',
 38: 'RPRE',
 39: 'VACT',
 40: 'VATT',
 41: 'VSTA',
 42: 'XVAE',
 43: 'XVAM',
 44: 'XVBB',
 45: 'XVBM',
 46: 'XVMM'}
```

```
# Inference
# ignore special tokens
text = 'จะว่าไปแล้วเชิงเทียนของผมก็สวยดีเหมือนกัน'
inputs = tokenizer(text, return_tensors="pt")
tokenized_input = tokenizer([text], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
print('tokens :', tokens)
with torch.no_grad():
    logits = model(**inputs).logits
predictions = torch.argmax(logits, dim=2)
predicted_token_class = [model.config.id2label[t.item()] for t in predictions[0]]
print('predict pos :', predicted_token_class)
```

```
tokens : ['<s>', '_', 'จะว่าไป', 'แล้ว', 'เชิง', 'เทียน', 'ของ', 'ผมก็', 'สวยดี', 'เหมือนกัน', '</s>']
predict pos : ['PUNC', 'PUNC', 'JSBR', 'JSBR', 'NCMN', 'NCMN', 'RPRE', 'JSBR', 'ADVN', 'ADVN', 'PUNC']
```

Evaluate model :

The output from the model is a softmax over classes. We choose the maximum class as the answer for evaluation. Again, we will ignore the -100 labels.

```
import pandas as pd
from IPython.display import display

def evaluation_report(y_true, y_pred, get_only_acc=False):
    # retrieve all tags in y_true
    tag_set = set()
    for sent in y_true:
        for tag in sent:
            tag_set.add(tag)
    for sent in y_pred:
        for tag in sent:
            tag_set.add(tag)
    tag_list = sorted(list(tag_set))

    # count correct points
    tag_info = dict()
    for tag in tag_list:
        tag_info[tag] = {'correct_tagged': 0, 'y_true': 0, 'y_pred': 0}

    all_correct = 0
    all_count = sum([len(sent) for sent in y_true])
    speacial_tag = 0
    for sent_true, sent_pred in zip(y_true, y_pred):
        for tag_true, tag_pred in zip(sent_true, sent_pred):
            # pass special token
            if tag_true == -100 :
                speacial_tag += 1
                pass
            if tag_true == tag_pred:
                tag_info[tag_true]['correct_tagged'] += 1
                all_correct += 1
            tag_info[tag_true]['y_true'] += 1
            tag_info[tag_pred]['y_pred'] += 1
    print('speacial_tag :',speacial_tag) # delete number of special token from all_count
    accuracy = (all_correct / (all_count-speacial_tag))

    # get only accuracy for testing
    if get_only_acc:
      return accuracy

    accuracy *= 100


    # summarize and make evaluation result
    eval_list = list()
    for tag in tag_list:
        eval_result = dict()
        eval_result['tag'] = tag
        eval_result['correct_count'] = tag_info[tag]['correct_tagged']
        precision = (tag_info[tag]['correct_tagged']/tag_info[tag]['y_pred'])*100 if tag_info[tag]['y_pred'] else '-'
        recall = (tag_info[tag]['correct_tagged']/tag_info[tag]['y_true'])*100 if (tag_info[tag]['y_true'] > 0) else 0
        eval_result['precision'] = precision
        eval_result['recall'] = recall
        eval_result['f1_score'] = (2*precision*recall)/(precision+recall) if (type(precision) is float and recall > 0) el

        eval_list.append(eval_result)

    eval_list.append({'tag': 'accuracy=%.2f' % accuracy, 'correct_count': '', 'precision': '', 'recall': '', 'f1_score':

    df = pd.DataFrame.from_dict(eval_list)
    df = df[['tag', 'precision', 'recall', 'f1_score', 'correct_count']]

    display(df)

```

```
# prepare test set
test_data = tokenized_orchid["test"]
```

```
# labels for test set
y_test = []
for inputs in test_data:
  y_test.append(inputs['labels'])
```

```
y_pred = []
device = 'cuda' if torch.cuda.is_available() else 'cpu'
for inputs in test_data:
    text = inputs['sentence']
    inputs = tokenizer(text, return_tensors="pt")
    with torch.no_grad():
        pred = model(**inputs).logits
```

```
        predictions = torch.argmax(pred, dim=2)
        # Append padded predictions to y_pred
        y_pred.append(predictions.tolist()[0])
```

```
# check our prediction with label
# -100 is special tokens : [<s>, </s>, _]
print(y_pred[0])
print(y_test[0])
```

```
[37, 29, 39, 26, 26, 26, 37, 37, 26, 26, 26, 41, 37, 37, 26, 26, 39, 26, 37]
[-100, 29, 39, 26, 26, 26, 37, -100, 26, 26, 26, 41, 37, -100, 26, 26, 39, 26, -100]
```

```
evaluation_report(y_test, y_pred)
```

```
speacial_tag : 21039
```

	tag	precision	recall	f1_score	correct_count
0	-100	-	0.0	-	0
1	0	-	0.0	-	0
2	1	69.254032	68.019802	68.631369	687
3	2	0.0	0.0	-	0
4	3	85.714286	10.344828	18.461538	6
5	4	85.714286	32.142857	46.753247	18
6	5	86.046512	21.387283	34.259259	37
7	6	55.775316	98.32636	71.176174	705
8	7	-	0.0	-	0
9	8	61.19403	72.84264	66.512167	287
10	10	76.292043	89.595376	82.410279	930
11	11	91.416309	93.421053	92.407809	426
12	12	67.1875	82.692308	74.137931	86
13	13	-	0.0	-	0
14	14	79.591837	75.728155	77.61194	78
15	15	87.987988	89.877301	88.92261	293
16	16	-	0.0	-	0
17	17	84.758364	91.935484	88.201161	228
18	18	69.768977	96.794872	81.089375	1057
19	19	-	0.0	-	0
20	20	100.0	58.823529	74.074074	10
21	21	91.994479	88.689288	90.311653	1333
22	22	70.754717	89.285714	78.947368	150
23	23	79.381443	81.052632	80.208333	77
24	24	95.357728	96.892342	96.11891	1746
25	25	83.167421	84.157509	83.659536	1838
26	26	86.697059	94.163685	90.276246	29477
27	27	76.826722	59.837398	67.276051	368
28	28	84.615385	75.862069	80.0	88
29	29	95.776772	98.755832	97.243492	635

Other Pretrained model

In this section, we will experiment by fine-tuning other pretrained models, such as airesearch/wangchanberta-base-wiki-newmm, to see how about their performance.

Since each model uses a different word-tokenization method. for example, **airesearch/wangchanberta-base-wiki-newmm uses newmm**, while **airesearch/wangchanberta-base-att-spm-uncased uses SentencePiece**. please try fine-tuning and compare the performance of these models.

#TODO 3

Choose Pretrained Model

```
model_names = [
    'airesearch/wangchanberta-base-att-spm-uncased',
    'airesearch/wangchanberta-base-wiki-newmm',
    'airesearch/wangchanberta-base-wiki-ssg',
    'airesearch/wangchanberta-base-wiki-sefr',
    'airesearch/wangchanberta-base-wiki-spm',
]

#@title Choose Pretrained Model
model_name = "airesearch/wangchanberta-base-wiki-newmm" #@

#create tokenizer
tokenizer = Tokenizer(model_name).from_pretrained(
                f'{model_name}',
                revision='main',
                model_max_length=416,)

```

model_name: airesearch/wangchanberta-base-wiki-newmm

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.py:795: FutureWarning: `resume_download` is deprec
  warnings.warn(
newmm.json: 100%  3.56M/3.56M [00:00<00:00, 17.7MB/s]
config.json: 100%  559/559 [00:00<00:00, 43.5kB/s]
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may
The tokenizer class you load from this checkpoint is 'RobertaTokenizer'.
The class this function is called from is 'ThaiWordsNewmmTokenizer'.
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may
The tokenizer class you load from this checkpoint is 'RobertaTokenizer'.
The class this function is called from is 'ThaiWordsNewmmTokenizer'.
```

```
example = orchidl["train"][1899]
print('sentence :', example["sentence"])
tokenized_input = tokenizer([example["sentence"]], is_split_into_words=True)
tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
print('tokens :',tokens)
print('label tokens :', example["label_tokens"])
```

```
sentence : โดยพิจารณาจากพจนานุกรมภาษาคู่ (bilingual transfer dictionary)
tokens : ['<s>', 'โดย', 'พิจารณา', 'จาก', 'พจนานุกรม', 'ภาษา', 'คู่', '<_>', '<unk>', '<_>', 'transfer', '<_>', 'dictionary'
label tokens : ['โดย', 'พิจารณา', 'จาก', 'พจนานุกรม', 'ภาษา', 'คู่', ' ', '(', 'bilingual transfer dictionary', ')']
```

It's the same problem as above.

Warning: Can we use same function as above ?

Warning: Please beware of <unk>, an unknown word token.

Warning: Please be careful of " ◌ำ ", the 'am' vowel. WangchanBERTa's internal preprocessing replaces all " ◌ำ " to " ◌' and 'า'

```
def majority_vote_pos(examples):

    ##########################################################################################################################
    # TO DO: Since the tokens from the output of the pretrained tokenizer
    # do not match the tokens in the label tokens of the dataset,
    # the task is to create a function to determine the POS tags of the tokens generated by the pretrained tokenizer.
    # This should be done by referencing the POS tags in the label tokens. If a token partially overlaps with others,
    # the POS tag from the segment with the greater number of characters should be assigned.
    #
    # Example :
    # "การประชุม" (9 chars) is formed from "การ" (3 chars) + "ประชุม" (6 chars).
    # "การ" has a POS tag of 21,
    # and "ประชุม" has a POS tag of 39.
    # Therefore, the POS tag for "การประชุม" is 39,
    # as "การประชุม" is derived more from the "ประชุม" part than from the "การ" part.
    #
    # 'ทางวิชาการ' (10 chars) is formed from 'ทาง' (3 chars) + 'วิชาการ' (7 chars)
    # "ทาง" has a POS tag of 26,
    # and "วิชาการ" has a POS tag of 2.
    # Therefore, the POS tag for "ทางวิชาการ" is 2,
    # as "ทางวิชาการ" is derived more from the "ทาง" part than from the "วิชาการ" part.

    # FILL CODE HERE
    # tokenize word by pretrained tokenizer
```

```
    tokenized_inputs = tokenizer([examples["sentence"]], is_split_into_words=True)
    label_tokens = examples["label_tokens"]
    pos_tags = examples["pos_tags"]
    new_pos_result = []

    new_tokens = tokenizer.convert_ids_to_tokens(tokenized_inputs["input_ids"])

    label_idx = 0
    i = 0

    # for (token, tag) in zip(label_tokens, pos_tags):
    #     print(f"'{token}': {tag}")
    # print("--------")

    try:
        for t in new_tokens:
            if t in ["<s>", "</s>", "▁"]:
                new_pos_result.append(-100)
                continue
            if t == "<unk>":
                new_pos_result.append(-100)
                # label_idx += 1
                continue

            buffer = ""
            weights = {}
            t = t.replace('ᵴ', 'ᵴ')
            t = t.replace("<_>", " ")
            if t[0] == "▁":
                t = t[1:]
            while label_tokens[label_idx][i] != t[0]:
                i += 1
                if i == len(label_tokens[label_idx]):
                    label_idx += 1
                    i = 0
            j = 1
            # print(f"BEGIN t: '{t}'")
            for a in range(30):
                buffer += label_tokens[label_idx][i]
                # print(f"buffer: {buffer}, t: {t[:j]}")

                if buffer != t[:j]:
                    buffer = ""
                    j = 1
                    continue

                j += 1
                if pos_tags[label_idx] not in weights:
                    weights[pos_tags[label_idx]] = 0
                weights[pos_tags[label_idx]] += 1
                i += 1
                if i == len(label_tokens[label_idx]):
                    label_idx += 1
                    i = 0
                if buffer == t:
                    break

            # for (token, tag) in zip(new_tokens, new_pos_result):
            #     print(f"'{token}': {tag}")
            # print(weights)
            max_key = max(weights, key=weights.get)
            new_pos_result.append(max_key)
    except IndexError:
        print(examples)
        for (token, tag) in zip(label_tokens, pos_tags):
            print(f"'{token}': {tag}")
        print("--------")
        for (token, tag) in zip(new_tokens, new_pos_result):
            print(f"'{token}': {tag}")
        # print(new_tokens, "\n", label_tokens, "\n", pos_tags)
        # print(new_pos_result)
        raise ValueError("Age cannot be negative!")

    tokenized_inputs['tokens'] = new_tokens
    tokenized_inputs['labels'] = new_pos_result

    return tokenized_inputs
    ###########################################################################################################################
```

```
tokenized_orchid = orchidl.map(majority_vote_pos)
```

```
Parameter 'function'=<function majority_vote_pos at 0x7bf5800cb600> of the transform datasets.arrow_dataset.Dataset._map
WARNING:datasets.fingerprint:Parameter 'function'=<function majority_vote_pos at 0x7bf5800cb600> of the transform datase
```

Map: 100% 18500/18500 [00:17<00:00, 1065.59 examples/s]

Map: 100% 4625/4625 [00:04<00:00, 1227.81 examples/s]

```python
# hard test case
example = tokenized_orchid["train"][1899]
for i in example :
    print(i, ":", example[i])
```

```
id : 1899
label_tokens : ['โดย', 'พิจารณา', 'จาก', 'พจนานุกรม', 'ภาษา', 'คู่', ' ', '(', 'bilingual transfer dictionary', ')']
pos_tags : [25, 39, 38, 26, 26, 5, 37, 37, 26, 37]
sentence : โดยพิจารณาจากพจนานุกรมภาษาคู่ (bilingual transfer dictionary)
input_ids : [0, 80, 3973, 45, 12252, 3496, 592, 5, 3, 5, 30055, 5, 63190, 178, 2]
token_type_ids : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
attention_mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
tokens : ['<s>', 'โดย', 'พิจารณา', 'จาก', 'พจนานุกรม', 'ภาษา', 'คู่', '<_>', '<unk>', '<_>', 'transfer', '<_>', 'dictionary'
labels : [-100, 25, 39, 38, 26, 26, 5, 37, -100, 26, 26, 26, 26, 37, -100]
```

Choose Pretrained Model

```python
model_names = [
    'wangchanberta-base-att-spm-uncased',
    'wangchanberta-base-wiki-newmm',
    'wangchanberta-base-wiki-ssg',
    'wangchanberta-base-wiki-sefr',
    'wangchanberta-base-wiki-spm',
]

#@title Choose Pretrained Model
model_name = "wangchanberta-base-wiki-newmm" #@param ["wan

#create model
model = AutoModelForTokenClassification.from_pretrained(
    f"airesearch/{model_name}",
    revision='main',
    num_labels=47, id2label=id2label, label2id=label2id
)

```

model_name: wangchanberta-base-wiki-newmm

pytorch_model.bin: 100% 646M/646M [00:06<00:00, 160MB/s]

```
/usr/local/lib/python3.11/dist-packages/transformers/modeling_utils.py:463: FutureWarning: You are using `torch.load` wi
  return torch.load(checkpoint_file, map_location="cpu")
Some weights of the model checkpoint at airesearch/wangchanberta-base-wiki-newmm were not used when initializing Roberta
- This IS expected if you are initializing RobertaForTokenClassification from the checkpoint of a model trained on anoth
- This IS NOT expected if you are initializing RobertaForTokenClassification from the checkpoint of a model that you exp
Some weights of RobertaForTokenClassification were not initialized from the model checkpoint at airesearch/wangchanberta
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```python
data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)
```

#TODO 4

Fine-tuning other pretrained model with our orchid corpus.

```python
training_args = TrainingArguments(
    ##########################
    output_dir="pos-base-wiki-newmm",
    learning_rate=2e-5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    num_train_epochs=2,
    weight_decay=0.01,
    push_to_hub=True
    ##########################
)

trainer = Trainer(
    ##########################
    model=model,
    args=training_args,
    train_dataset=tokenized_orchid["train"],
    eval_dataset=tokenized_orchid["test"],
```

```
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    ########################
)

trainer.train()
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_deprecation.py:131: FutureWarning: 'Repository' (from 'hu
For more details, please read https://huggingface.co/docs/huggingface_hub/concepts/git_vs_http.
  warnings.warn(warning_message, FutureWarning)
Cloning https://huggingface.co/Jirayuwat12/pos-base-wiki-newmm into local empty directory.
WARNING:huggingface_hub.repository:Cloning https://huggingface.co/Jirayuwat12/pos-base-wiki-newmm into local empty direc
/usr/local/lib/python3.11/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation of AdamW is
  warnings.warn(
```

[1158/1158 07:57, Epoch 2/2]

Step	Training Loss
500	0.592600
1000	0.302100

```
TrainOutput(global_step=1158, training_loss=0.42407236774556584, metrics={'train_runtime': 477.395,
'train_samples_per_second': 77.504, 'train_steps_per_second': 2.426, 'total_flos': 912923649812664.0, 'train_loss':
0.42407236774556584, 'epoch': 2.0})
```

```
######## EVALUATE YOUR MODEL ########
from transformers import AutoModelForTokenClassification

## Load your fine-tuned model from Hugging Face
model = AutoModelForTokenClassification.from_pretrained("jirayuwat12/pos-base-wiki-newmm") ## your model path from Huggir

# prepare test set
test_data = tokenized_orchid["test"]

# labels for test set
y_test = []
for inputs in test_data:
  y_test.append(inputs['labels'])
y_pred = []
device = 'cuda' if torch.cuda.is_available() else 'cpu'
for inputs in test_data:
    text = inputs['sentence']
    inputs = tokenizer(text, return_tensors="pt")
    with torch.no_grad():
        pred = model(**inputs).logits
        predictions = torch.argmax(pred, dim=2)
        # Append padded predictions to y_pred
        y_pred.append(predictions.tolist()[0])

######## EVALUATE YOUR MODEL ########
evaluation_report(y_test, y_pred)
```

speacial_tag : 11485

	tag	precision	recall	f1_score	correct_count
0	-100	-	0.0	-	0
1	0	10.0	6.666667	8.0	1
2	1	76.66999	69.845595	73.098859	769
3	2	50.0	0.877193	1.724138	1
4	3	45.16129	25.0	32.183908	14
5	4	94.230769	79.032258	85.964912	49
6	5	78.030303	59.537572	67.540984	103
7	6	77.995643	91.094148	84.037559	358
8	7	-	0.0	-	0
9	8	70.241287	71.389646	70.810811	262
10	10	91.971665	88.422247	90.162037	779
11	11	93.461538	92.395437	92.92543	486

#TODO 5

Compare the results between both models. Are they comparable? (Think about the ground truths of both models).

Propose a way to fairly evaluate the models.

Write your answer here :

It's not camparable because each model is trained using different dataset and different size

there are many approaches to make a fair comparision e.g.