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

```
!pip install --upgrade transformers datasets evaluate seqeval accelerate bitsandbytes spacy
```

Show hidden output

```
import os
import json
import torch
import pandas as pd
import spacy

from huggingface_hub import login
from datasets import Dataset
from transformers import (
    AutoTokenizer,
    AutoModelForTokenClassification
)
```

```
# ── STEP 2: Authenticate once (so you won't see HF_TOKEN warnings) ──────────
login("hf_token")
```

```
# ── STEP 3: Load your CSV & rename columns to our canonicals ────────────────
df = pd.read_csv("open_ave_data.csv")
df = df.rename(columns={
    "ReportText":     "text",
    "ExamName":       "Examination",
    "clinicaldata":   "Clinical",
    "findings":       "Findings",
    "impression":     "Impression",
})
print("Sample report:", df["text"].iloc[0])
```

```
Sample report: EXAM: CHEST RADIOGRAPHY EXAM DATE: 06/01/2019 08:30 PM. CLINICAL HISTORY: Cough. COMPARISON: None. TECHNIQUE: 2 views. FINDINGS: Lungs/Pleura: No focal opacities evider
```

```
# ── STEP 4: Build BIOES-labeled JSONL from scratch ──────────────────────────
nlp = spacy.blank("en")  # whitespace tokenizer preserving char offsets
label_data = []

for _, row in df.iterrows():
    text   = row["text"]
    doc    = nlp(text)
    tokens = [tok.text for tok in doc]
    labels = ["O"] * len(tokens)

    for field in ["Examination","Clinical","Findings","Impression"]:
        span = row[field]
        if not isinstance(span, str) or not span.strip():
            continue
        start = text.find(span)
        if start < 0:
            continue
        end = start + len(span)
        tok_idxs = [i for i, t in enumerate(doc)
                    if not (t.idx + len(t.text) <= start or t.idx >= end)]
        if not tok_idxs:
            continue

        if len(tok_idxs) == 1:
            labels[tok_idxs[0]] = f"S-{field}"
        else:
            for j, ti in enumerate(tok_idxs):
                if   j == 0:               labels[ti] = f"B-{field}"
                elif j == len(tok_idxs)-1: labels[ti] = f"E-{field}"
                else:                      labels[ti] = f"I-{field}"

    label_data.append({"tokens": tokens, "labels": labels})

with open("labeled_data.jsonl", "w") as fout:
    for ex in label_data:
        fout.write(json.dumps(ex) + "\n")
print(f"Wrote {len(label_data)} examples → labeled_data.jsonl")
```

```
Wrote 954 examples → labeled_data.jsonl
```

```
# ── STEP 5: Load JSONL as a Hugging Face Dataset ────────────────────────────
ds = Dataset.from_json("labeled_data.jsonl")
print(ds[0])
```

```
, ':', 'Cough', '.', 'COMPARISON', ':', 'None', '.', 'TECH
```

```
# ── STEP 6: Load the QWen 0.8 B tokenizer & quick sanity-check ──────────────
MODEL_NAME = "Qwen/Qwen3-0.6B"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, trust_remote_code=True)
sample = tokenizer(ds[0]["tokens"], is_split_into_words=True)
print("token_ids:", sample["input_ids"][:10])
```

```
/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), set it as secret in your Google Colab and restart your sessior
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(
tokenizer_config.json:  9.73k/?  [00:00<00:00,  668kB/s]
vocab.json:  2.78M/?  [00:00<00:00,  2.90MB/s]
merges.txt:  1.67M/?  [00:00<00:00,  6.07MB/s]
tokenizer.json:  100%  11.4M/11.4M  [00:00<00:00,  65.7kB/s]
token_ids: [3257, 1402, 25, 43793, 784, 49, 42853, 39984, 56, 3257]
```

```
# ── STEP 7: Define BIOES tags & mappings ────────────────────────────────────
fields     = ["Examination", "Clinical", "Findings", "Impression"]
bioes      = [f"{pfx}-{fld}" for fld in fields for pfx in ["B", "I", "E", "S"]]
label_list = bioes + ["O"]
label2id   = {lab: i for i, lab in enumerate(label_list)}
id2label   = {i: lab for lab, i in label2id.items()}
```

```
# ── STEP 8: Tokenize & align BIOES labels to word-pieces (robust version) ───
def tokenize_and_align_labels(examples):
```

```
    tokenized = tokenizer(
        examples["tokens"],
        is_split_into_words=True,
        truncation=True,
        max_length=512
    )
    all_labels = []
    for i, label_seq in enumerate(examples["labels"]):
        word_ids    = tokenized.word_ids(batch_index=i)
        prev_wid    = None
        aligned_lbl = []
        for wid in word_ids:
            # 1) if no word or out of range, O
            if wid is None or wid < 0 or wid >= len(label_seq):
                aligned_lbl.append("O")
            else:
                lbl = label_seq[wid]
                # 2) if this is continuation of same word, force I-*
                if wid == prev_wid:
                    if lbl.startswith("B-") or lbl.startswith("S-"):
                        base = lbl.split("-",1)[1]
                        lbl  = "I-" + base
                aligned_lbl.append(lbl)
            prev_wid = wid
        all_labels.append(aligned_lbl)

    tokenized["labels"] = all_labels
    return tokenized

# Apply it safely
tokenized_ds = ds.map(tokenize_and_align_labels, batched=True)
print(tokenized_ds[0])
```

```
Map: 100%  954/954 [00:00<00:00, 999.86 examples/s]
{'tokens': ['EXAM', ':', 'CHEST', 'RADIOGRAPHY', 'EXAM', 'DATE', ':', '06/01/2019', '08:30', 'PM', '.', 'CLINICAL', 'HISTORY', ':', 'Cough', '.', 'COMPARISON', ':', 'None', '.', 'TECH
```

```
# ── STEP 9: Convert string labels → integer IDs ──────────────────────
def encode_label_ids(batch):
    batch["labels"] = [
        [ label2id.get(l, label2id["O"]) for l in labs ]
        for labs in batch["labels"]
    ]
    return batch

tokenized_ds = tokenized_ds.map(encode_label_ids, batched=True)
```

```
Map: 100%  954/954 [00:00<00:00, 5910.92 examples/s]
```

```
# ── STEP 10: Split into train & validation ──────────────────────────
from datasets import DatasetDict
split = tokenized_ds.train_test_split(test_size=0.1, seed=42)
split_ds = DatasetDict(train=split["train"], validation=split["test"])
```

```
# ── STEP 11: Load the token-classification model (QWen 0.6B) ────────
from transformers import AutoModelForTokenClassification

model = AutoModelForTokenClassification.from_pretrained(
    "Qwen/Qwen3-0.6B",
    num_labels=len(label_list),
    id2label=id2label,
    label2id=label2id,
    trust_remote_code=True
)
```

```
config.json: 100%  726/726 [00:00<00:00, 46.4kB/s]
model.safetensors: 100%  1.50G/1.50G [00:46<00:00, 23.7MB/s]
Some weights of Qwen3ForTokenClassification were not initialized from the model checkpoint at Qwen/Qwen3-0.6B and are newly initialized: ['score.bias', 'score.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
# ── STEP 12a: Prepare metric and compute function ──────────────────
import evaluate, numpy as np
seqeval = evaluate.load("seqeval")

def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=-1)
    labs  = p.label_ids
    true_labels = [
        [id2label[l] for l in lr if l != -100]
        for lr in labs
    ]
    true_preds = [
        [id2label[p_] for p_, l in zip(pr, lr) if l != -100]
        for pr, lr in zip(preds, labs)
    ]
    res = seqeval.compute(predictions=true_preds, references=true_labels)
    return {
        "precision": res["overall_precision"],
        "recall":    res["overall_recall"],
        "f1":        res["overall_f1"],
        "accuracy":  res["overall_accuracy"],
    }
```

```
Downloading builder script: 6.34k/? [00:00<00:00, 167kB/s]
```

```
# ── STEP 12b: Data collator & training arguments ────────────────────
from transformers import DataCollatorForTokenClassification, TrainingArguments, Trainer

data_collator = DataCollatorForTokenClassification(tokenizer)

args = TrainingArguments(
    output_dir="qwen_finetuned_qwen3-0.6B",
    per_device_train_batch_size=4,
    per_device_eval_batch_size=8,
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=5e-5,
    num_train_epochs=3,
    fp16=True,
    logging_dir="logs"
)
```

```
# ── STEP 12c: Initialize the Trainer ────────────────────────────────
trainer = Trainer(
```

```
    model=model,
    args=args,
    train_dataset=split_ds["train"],
    eval_dataset=split_ds["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

```
/tmp/ipython-input-1534567402.py:2: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
```

```
# ── STEP 13: Train, Evaluate & Save ──────────────────────────────────────────
trainer.train()
metrics = trainer.evaluate()
print("Evaluation results:", metrics)

trainer.save_model("qwen_finetuned_qwen3-0.6B")
```

```
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run name by setting the `Trair
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize?ref=models
wandb: Paste an API key from your profile and hit enter: ··········
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: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: suhagadge (suhagadge-iit) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
```

Tracking run with wandb version 0.21.0
Run data is saved locally in `/content/wandb/run-20250804_194953-kyu96y3t`
Syncing run **qwen_finetuned_qwen3-0.6B** to Weights & Biases (docs)
View project at https://wandb.ai/suhagadge-iit/huggingface
View run at https://wandb.ai/suhagadge-iit/huggingface/runs/kyu96y3t
[645/645 17:49, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	No log	0.121378	0.868613	0.868613	0.868613	0.984909
2	No log	0.114620	0.902985	0.883212	0.892989	0.986274
3	0.119700	0.086638	0.875000	0.868613	0.871795	0.987237

```
/usr/local/lib/python3.11/dist-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sample
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.11/dist-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sample
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.11/dist-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sample
  _warn_prf(average, modifier, msg_start, len(result))
```

[12/12 00:01]

```
/usr/local/lib/python3.11/dist-packages/seqeval/metrics/v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted sample
  _warn_prf(average, modifier, msg_start, len(result))
Evaluation results: {'eval_loss': 0.08663821220397949, 'eval_precision': 0.875, 'eval_recall': 0.8686131386861314, 'eval_f1': 0.8717948717948718, 'eval_accuracy': 0.9872371167121529,
```

```
from transformers import pipeline

pipe = pipeline(
    "token-classification",
    model="qwen_finetuned_qwen3-0.6B",
    tokenizer=tokenizer,
    aggregation_strategy="simple"
)

example = df["text"].iloc[0]
print(pipe(example))
```

```
Device set to use cuda:0
[{'entity_group': 'Impression', 'score': np.float32(0.99920416), 'word': ' IMPRESSION: Normal 2-view chest radiography', 'start': 299, 'end': 343}, {'entity_group': 'Impression', 'scc
```