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

```
  Building wheel for seqeval (setup.py) ... done
  Created wheel for seqeval: filename=seqeval-1.2.2-py3-none-any.whl size=16162 sha256=2bdbd50815bf4ac16d3002401a09140f2f520efd309e471dd57bd6839a65ddf5
  Stored in directory: /root/.cache/pip/wheels/bc/92/f0/243288f899c2eacdfa8c5f9aede4c71a9bad0ee26a01dc5ead
Successfully built seqeval
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-nccl-cu12
    Found existing installation: nvidia-nccl-cu12 2.23.4
    Uninstalling nvidia-nccl-cu12-2.23.4:
      Successfully uninstalled nvidia-nccl-cu12-2.23.4
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
  Attempting uninstall: transformers
    Found existing installation: transformers 4.54.1
    Uninstalling transformers-4.54.1:
      Successfully uninstalled transformers-4.54.1
  Attempting uninstall: accelerate
    Found existing installation: accelerate 1.9.0
    Uninstalling accelerate-1.9.0:
      Successfully uninstalled accelerate-1.9.0
Successfully installed accelerate-1.10.0 bitsandbytes-0.46.1 evaluate-0.4.5 nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-r
```

```
import os, gc, json, math
import torch
import pandas as pd
import spacy

from datasets import load_dataset, Dataset, concatenate_datasets
from transformers import (
    AutoTokenizer,
    AutoModelForTokenClassification,
    DataCollatorForTokenClassification,
    TrainingArguments,
    Trainer,
    TrainerCallback,
)
from seqeval.metrics import classification_report
import evaluate
```

```
df = pd.read_csv("open_ave_data.csv")
print("Raw columns:", df.columns.tolist())
df.head()
```

```
Raw columns: ['Unnamed: 0', 'ReportText', 'findings', 'clinicaldata', 'ExamName', 'impression']
```

	Unnamed: 0	ReportText	findings	clinicaldata	ExamName	impression
0	0	EXAM: CHEST RADIOGRAPHY EXAM DATE: 06/01/2019 ...	FINDINGS: Lungs/Pleura: No focal opacities evi...	CLINICAL HISTORY: Cough. \n\n	EXAM: CHEST RADIOGRAPHY EXAM DATE: 06/01/2019 ...	IMPRESSION: Normal 2-view chest radiography.
1	1	EXAM: CHEST RADIOGRAPHY EXAM DATE: 05/23/2020 ...	FINDINGS: Lungs/Pleura: No focal opacities evi...	CLINICAL HISTORY: CHEST PAIN. \n\n	EXAM: CHEST RADIOGRAPHY EXAM DATE: 05/23/2020 ...	IMPRESSION: No acute cardiopulmonary abnormali...
2	2	EXAM: CHEST RADIOGRAPHY EXAM DATE: 12/13/2019 ...	FINDINGS: Lungs/Pleura: No focal opacities evi...	CLINICAL HISTORY: CHEST PAIN. \n\n	EXAM: CHEST RADIOGRAPHY EXAM DATE: 12/13/2019 ...	IMPRESSION: No acute cardiopulmonary process.
3	3	Exam: - CHEST-PORTABLE History: Chest pain Com...	Findings: Heart size appears normal. Lungs cle...	History: Chest pain \n\n	Exam: - CHEST-PORTABLE\n\nComparison: None	Impression: Lungs clear
4	4	EXAM: CHEST RADIOGRAPHY EXAM DATE: 06/17/2021 ...	FINDINGS: Lungs/Pleura: No focal opacities evi...	CLINICAL HISTORY: CHEST PAIN, SHORTNESS OF BRE...	EXAM: CHEST RADIOGRAPHY EXAM DATE: 06/17/2021 ...	IMPRESSION: Normal single view chest.

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# 1) Drop duplicates & fully empty rows
df_clean = df.drop_duplicates().dropna(how="all").copy()

# 2) Trim whitespace in string columns
for c in df_clean.select_dtypes(include="object").columns:
    df_clean[c] = df_clean[c].str.strip()

# 3) Fill numeric NaNs with medians
for c in df_clean.select_dtypes(include="number").columns:
    df_clean[c] = df_clean[c].fillna(df_clean[c].median())

print("Cleaned shape:", df_clean.shape)
df_clean.head()
```

```
Cleaned shape: (954, 6)
```

	Unnamed: 0	ReportText	findings	clinicaldata	ExamName	impression
0	0	EXAM: CHEST RADIOGRAPHY EXAM DATE: 06/01/2019 ...	FINDINGS: Lungs/Pleura: No focal opacities evi...	CLINICAL HISTORY: Cough.	EXAM: CHEST RADIOGRAPHY EXAM DATE: 06/01/2019 ...	IMPRESSION: Normal 2-view chest radiography.
1	1	EXAM: CHEST RADIOGRAPHY EXAM DATE: 05/23/2020 ...	FINDINGS: Lungs/Pleura: No focal opacities evi...	CLINICAL HISTORY: CHEST PAIN.	EXAM: CHEST RADIOGRAPHY EXAM DATE: 05/23/2020 ...	IMPRESSION: No acute cardiopulmonary abnormality.
2	2	EXAM: CHEST RADIOGRAPHY EXAM DATE: 12/13/2019 ...	FINDINGS: Lungs/Pleura: No focal opacities evi...	CLINICAL HISTORY: CHEST PAIN.	EXAM: CHEST RADIOGRAPHY EXAM DATE: 12/13/2019 ...	IMPRESSION: No acute cardiopulmonary process.
3	3	Exam: - CHEST-PORTABLE History: Chest pain Com...	Findings: Heart size appears normal. Lungs clear.	History: Chest pain	Exam: - CHEST-PORTABLE\n\nComparison: None	Impression: Lungs clear
4	4	EXAM: CHEST RADIOGRAPHY EXAM DATE: 06/17/2021 ...	FINDINGS: Lungs/Pleura: No focal opacities evi...	CLINICAL HISTORY: CHEST PAIN, SHORTNESS OF BRE...	EXAM: CHEST RADIOGRAPHY EXAM DATE: 06/17/2021 ...	IMPRESSION: Normal single view chest.

Next steps: Generate code with df_clean | View recommended plots | New interactive sheet

```
df_clean.to_csv("cleaned_data.csv", index=False)
```

```
# — Step 4 (revised): Build BIOES JSONL with robust Title extraction —
import re, json
import spacy
import pandas as pd
from difflib import SequenceMatcher

nlp = spacy.blank("en")

dfc = pd.read_csv("cleaned_data.csv")

# Identify text column & rename GT columns
report_col = "ReportText" if "ReportText" in dfc.columns else dfc.columns[0]
dfc = dfc.rename(columns={
    "ExamName": "Title",
    "clinicaldata": "Clinical_Indication",
    "findings": "Findings",
    "impression": "Impression"
})

# --- helpers ---
FIELD_STOPWORDS = r"(?:TECHNIQUE|INDICATION|HISTORY|CLINICAL|COMPARISON|FINDINGS|IMPRESSION|CONCLUSION|RECOMMENDATIONS)"
TITLE_PAT = re.compile(
    rf"(?:^|\n)\s*(EXAM(?:INATION)?|STUDY|PROCEDURE)\s*[:\-]?\s*(.+?)(?=(?:\n|\. |\r|{FIELD_STOPWORDS}\b))",
    flags=re.IGNORECASE | re.DOTALL
)

def norm(s: str) -> str:
    return re.sub(r"[^a-z0-9]+", " ", str(s).lower()).strip()

def best_fuzzy_span(text: str, target: str, window=8):
    """
    Slide a window over whitespace tokens and pick the fragment with max similarity to target.
    Returns (start_char, end_char) or (None, None).
    """
    if not target:
        return (None, None)
    words = str(text).split()
    tgt_n = norm(target)
    if not tgt_n:
        return (None, None)

    best_score, best_span = 0.0, (None, None)
    for i in range(len(words)):
        for j in range(i+1, min(len(words), i+window)+1):
            frag = " ".join(words[i:j]).strip()
            if not frag:
                continue
            score = SequenceMatcher(None, norm(frag), tgt_n).ratio()
            if score > best_score:
                # find exact char offsets for this frag (case-insensitive)
                m = re.search(re.escape(frag), text, flags=re.IGNORECASE)
                if m:
                    best_score, best_span = score, (m.start(), m.end())

    return best_span if best_score >= 0.6 else (None, None)

def find_title_span(text: str, examname: str):
    # 1) Regex on EXAM/STUDY/PROCEDURE lines
    m = TITLE_PAT.search(text)
    if m:
        start = m.start(2)
        end   = start + len(m.group(2).strip())
        return (start, end)
    # 2) Fallback: fuzzy match ExamName into text
    return best_fuzzy_span(text, examname or "")

# --- build BIOES ---
label_data = []
title_hits = 0

for _, row in dfc.iterrows():
    text = str(row.get(report_col, "") or "")
    doc  = nlp(text)
    tokens = [t.text for t in doc]
    labels = ["O"] * len(tokens)

    def tag_span(start, end, field):
        if start is None or end is None:
            return False
        idxs = [i for i, t in enumerate(doc)
                if not (t.idx + len(t.text) <= start or t.idx >= end)]
        if not idxs:
            return False
        if len(idxs) == 1:
            labels[idxs[0]] = f"S-{field}"
        else:
            for j, ti in enumerate(idxs):
                if j == 0:
                    labels[ti] = f"B-{field}"
                elif j == len(idxs) - 1:
                    labels[ti] = f"E-{field}"
                else:
                    labels[ti] = f"I-{field}"
        return True

    # Title via regex/fuzzy
    t_start, t_end = find_title_span(text, str(row.get("Title", "")))
    if tag_span(t_start, t_end, "Title"):
        title_hits += 1
```

```
    # Other fields (case-insensitive exact match is usually fine)
    for field in ["Clinical_Indication", "Findings", "Impression"]:
        span = row.get(field, "")
        if not isinstance(span, str) or not span.strip():
            continue
        start = text.lower().find(span.lower())
        if start < 0:
            continue
        end = start + len(span)
        tag_span(start, end, field)

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

with open("labeled_data.jsonl", "w", encoding="utf-8") as f:
    for ex in label_data:
        f.write(json.dumps(ex) + "\n")

print(f"Wrote {len(label_data)} → labeled_data.jsonl  | Title spans found: {title_hits}")
```

```
Wrote 954 → labeled_data.jsonl  | Title spans found: 943
```

```
from collections import Counter, defaultdict, deque
cnt = Counter()
for ex in label_data:
    cnt.update(ex["labels"])
print("Label frequencies (top 20):", cnt.most_common(20))
```

```
Label frequencies (top 20): [('I-Findings', 30486), ('I-Impression', 14793), ('O', 12010), ('I-Title', 5128), ('I-Clinical_Indication', 4508), ('B-Findings', 950), ('E-Findings', 950)
```

```
from datasets import load_dataset, concatenate_datasets

raw_ds = load_dataset("json", data_files="labeled_data.jsonl", split="train")

# Split for stratification only (no duplication)
has_title = raw_ds.filter(lambda ex: any(l.startswith("B-Title") for l in ex["labels"]))
no_title  = raw_ds.filter(lambda ex: not any(l.startswith("B-Title") for l in ex["labels"]))

print(f"Counts → Title-rich: {len(has_title)}, Title-free: {len(no_title)}")

# Independent splits, then recombine → ensures Title appears in eval
eval_frac = 0.10
split_t = has_title.train_test_split(test_size=eval_frac, seed=42)
split_n = no_title.train_test_split(test_size=eval_frac, seed=42)

train_ds = concatenate_datasets([split_t["train"], split_n["train"]]).shuffle(seed=42)
eval_ds  = concatenate_datasets([split_t["test"],  split_n["test"]]).shuffle(seed=42)

print(f"→ train: {len(train_ds)}   eval: {len(eval_ds)}")
```

```
Generating train split:        954/0  [00:00<00:00,  12321.56  examples/s]

Filter: 100%                                                    954/954  [00:00<00:00,  3295.05  examples/s]

Filter: 100%                                                    954/954  [00:00<00:00,  3197.68  examples/s]

Counts → Title-rich: 906, Title-free: 48
→ train: 858   eval: 96
```

```
from transformers import AutoTokenizer

labels = [
  "O",
  "B-Title","I-Title","E-Title","S-Title",
  "B-Clinical_Indication","I-Clinical_Indication","E-Clinical_Indication","S-Clinical_Indication",
  "B-Findings","I-Findings","E-Findings","S-Findings",
  "B-Impression","I-Impression","E-Impression","S-Impression"
]
id2label = {i: lab for i, lab in enumerate(labels)}
label2id = {lab: i for i, lab in id2label.items()}

MODEL_NAME = "Qwen/Qwen3-0.6B"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, trust_remote_code=True)
```

```
/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 session
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,  366kB/s]

vocab.json:        2.78M/?  [00:00<00:00,  10.2MB/s]

merges.txt:        1.67M/?  [00:00<00:00,  27.7MB/s]

tokenizer.json:  100%                                           11.4M/11.4M  [00:00<00:00,  48.1kB/s]
```

```
from transformers import DataCollatorForTokenClassification

def tokenize_and_align_labels(examples):
    tok = tokenizer(
        examples["tokens"],
        is_split_into_words=True,
        truncation=True,
        max_length=128
    )
    word_ids = tok.word_ids()
    aligned = []
    for wid in word_ids:
        aligned.append(-100 if wid is None else label2id[examples["labels"][wid]])
    tok["labels"] = aligned
    return tok

train_tok = train_ds.map(tokenize_and_align_labels, batched=False)
eval_tok  = eval_ds.map(tokenize_and_align_labels,  batched=False)

data_collator = DataCollatorForTokenClassification(tokenizer)
print("✅ Tokenized:", train_tok.shape, eval_tok.shape)
```

```
Map: 100%                                                       858/858  [00:01<00:00,  448.24  examples/s]

Map: 100%                                                       96/96  [00:00<00:00,  606.87  examples/s]

✅ Tokenized: (858, 4) (96, 4)
```

```
import evaluate
from seqeval.metrics import classification_report

seqeval = evaluate.load("seqeval")

def compute_metrics(p):
    import numpy as np
    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)]

    overall = seqeval.compute(predictions=true_preds, references=true_labels)
    report  = classification_report(true_labels, true_preds,
                                    output_dict=True, digits=4, zero_division=0)

    per_ent = {}
    for ent in ["Title","Clinical_Indication","Findings","Impression"]:
        per_ent[f"{ent}_prec"] = report.get(ent, {}).get("precision", 0.0)
        per_ent[f"{ent}_rec"]  = report.get(ent, {}).get("recall",    0.0)
        per_ent[f"{ent}_f1"]   = report.get(ent, {}).get("f1-score",  0.0)

    return {
        "overall_precision": overall.get("overall_precision", 0.0),
        "overall_recall":    overall.get("overall_recall",    0.0),
        "overall_f1":        overall.get("overall_f1",        0.0),
        "overall_accuracy":  overall.get("overall_accuracy",  0.0),
        **per_ent
    }
```

```
Downloading builder script:        6.34k/?  [00:00<00:00,  345kB/s]
```

```
from transformers import TrainerCallback

class CSVLogger(TrainerCallback):
    def __init__(self, path="metrics.csv"):
        import csv
        self.fp = open(path, "w", newline="")
        self.w  = csv.writer(self.fp)
        self.header_written = False

    def on_evaluate(self, args, state, control, metrics=None, **kwargs):
        if not self.header_written:
            self.w.writerow(["step"] + list(metrics.keys()))
            self.header_written = True
        self.w.writerow([state.global_step] + [metrics[k] for k in metrics])
        self.fp.flush()
```

```
import gc, torch
from transformers import AutoModelForTokenClassification, TrainingArguments, Trainer

# Clean up any leftovers (optional)
for name in ["trainer","model"]:
    globals().pop(name, None)
gc.collect(); torch.cuda.empty_cache()

# 1) Model
model = AutoModelForTokenClassification.from_pretrained(
    MODEL_NAME,
    num_labels=len(labels),
    id2label=id2label,
    label2id=label2id,
    trust_remote_code=True
)
model.gradient_checkpointing_enable()  # reduces VRAM at the cost of compute

# 2) Training args (T4 safe)
training_args = TrainingArguments(
    output_dir="qwen3-medex-finetuned",
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="overall_f1",

    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    gradient_accumulation_steps=4,   # effective batch ~4

    num_train_epochs=3,
    learning_rate=2e-5,
    warmup_ratio=0.1,
    max_grad_norm=1.0,

    fp16=True,
    optim="adamw_torch",

    logging_dir="logs",
    logging_steps=50,
    report_to=[],   # no WandB/TensorBoard
)

# 3) Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_tok,
    eval_dataset=eval_tok,
    data_collator=data_collator,
    tokenizer=tokenizer,          # deprecation warning is fine
    compute_metrics=compute_metrics,
    callbacks=[CSVLogger("metrics.csv")]
)

# 4) Train → Evaluate → Save
trainer.train()
final_metrics = trainer.evaluate()
print("Final metrics:", final_metrics)

trainer.save_model("qwen3-medex-finetuned/best")
```

```
config.json: 100%                                   726/726 [00:00<00:00, 70.8kB/s]

model.safetensors: 100%                             1.50G/1.50G [00:14<00:00, 174MB/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.
/tmp/ipython-input-3970947473.py:45: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
`use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=False`.
[645/645 29:58, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss	Overall Precision	Overall Recall	Overall F1	Overall Accuracy	Title Prec	Title Rec	Title F1	Clinical Indication Prec	Clinical Indication Rec	Clinical Indication F1	Findings Prec	Findings Rec	Findings F1	Impression Prec	Impressic Rec
1	0.765500	0.087460	0.821429	0.907554	0.862346	0.975800	0.578947	0.798883	0.671362	0.786477	0.850000	0.817006	0.967509	0.981685	0.974545	0.988571	0.98857
2	0.365900	0.068205	0.855932	0.910936	0.882578	0.978733	0.663551	0.793296	0.722646	0.857143	0.876923	0.866920	0.960573	0.981685	0.971014	0.918919	0.97142
3	0.347300	0.054102	0.854430	0.913191	0.882834	0.982308	0.683962	0.810056	0.741688	0.797153	0.861538	0.828096	0.956989	0.978022	0.967391	0.988636	0.99428

```
[96/96 00:05]
Final metrics: {'eval_loss': 0.054101813584566116, 'eval_overall_precision': 0.8544303797468354, 'eval_overall_recall': 0.9131905298759865, 'eval_overall_f1': 0.8828337874659401, 'eva
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.