Nya-Hoba NER — Colab Notebook

Named Entity Recognition for Low-Resource African Languages: A Transformer-Based Case Study on Nya-Hoba

Owner: Chahyaandida Ishaya

This notebook is prepared to run in Google Colab. Before running heavy training cells, set `Runtime -> Change runtime type -> GPU`.

Project Objectives

- **General Objective:** Design and evaluate a transformer-based NER system for Nya-Hoba.
- **Specific Objectives:**
  1. Collect, clean, and annotate a Nya-Hoba text corpus for NER tasks.
  2. Develop baseline NER models using traditional machine learning approaches for benchmarking.
  3. Fine-tune transformer-based models for NER on Nya-Hoba.
  4. Evaluate model performance using Precision, Recall, and F1-score.
  5. Release an open-source dataset and pre-trained models.

1. Setup

Run the code cell to install required packages. On Colab this may take a few minutes.

You may want to manually install a CUDA-compatible `torch` build if you intend to use GPU acceleration.

```
# Install dependencies
!pip install -q scikit-learn sklearn-crfsuite pandas joblib transformers datasets seqeval torch
print('Installed packages. Check versions:')
import sklearn, pandas, joblib, transformers
import torch
print('sklearn', sklearn.__version__)
print('pandas', pandas.__version__)
print('transformers', transformers.__version__)
print('torch', torch.__version__, 'CUDA:', torch.cuda.is_available())
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.6/43.6 kB 2.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 40.7 MB/s eta 0:00:00
  Building wheel for seqeval (setup.py) ... done
Installed packages. Check versions:
sklearn 1.6.1
pandas 2.2.2
transformers 4.56.1
torch 2.8.0+cu126 CUDA: True
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

2. Data Generation & Cleaning

This stage expands small seed lists of entity words (names, times, animals, and locations) into larger synthetic datasets through controlled randomization and pattern-based augmentation.

```
import json
import random
import os

#Making directory for data
os.makedirs('/content/data', exist_ok=True)
print('Your Data is located in /content/data/dataset.conll')

# Define the drive path
drive_path = '/content/drive/MyDrive/NER_nyahoba/notebook/content/data'
```

```
# ---------------------------
# STEP 1. Fixed lists
# ---------------------------
time_words = [
    "Zəkəu", "Sakana", "əna", "əhna", "Pəshinda", "Pishinda",
    "Fer pəchingə zekeu", "Pər pəchingə zekeu", "Pəchi", "Hya"
]

animal_words = [
    "Kwa", "Mabəlang", "Təga", "Gwanba", "Ha'l", "Dləgwam", "Thla",
    "Kətən", "Chiwar", "Ləvari", "Mapəla'u", "Litsa"
]

person_words_fixed = [
    "Chahyaandida", "Chabiya", "Hyellama", "Hyelnaya", "Wandiya", "Hyel", "Yesu",
    "Chataimada", "Chatramada", "Nanunaya", "Mapida", "Shimbal", "Chai",
    "Hyellachardati", "Hyellachardati", "Wamanyi", "Miyaninyi", "Miyakindahyelni", "Miyaninyi"
]

# ---------------------------
# STEP 2. Synthetic PERSON names
# ---------------------------
base_names = [
    "Abubakar", "Ibrahim", "Musa", "Usman", "Kabiru", "Bello", "Suleiman",
    "Ahmad", "Aliyu", "Shehu", "Aminu", "Habiba", "Fatima", "Aisha", "Zainab", "Hauwa",
    "Ruqayya", "Maryam", "Khadija", "Sa'adatu", "Yakubu", "Ismaila", "Nasiru", "Idris",
    "John", "Paul", "Peter", "James", "Joseph", "Stephen", "Samuel",
    "David", "Daniel", "Thomas", "Andrew", "Philip", "Simon", "Nathaniel",
    "Grace", "Joyce", "Ruth", "Esther", "Naomi", "Sarah", "Deborah",
    "Ndyako", "Pwakina", "Gargam", "Kwada", "Tizhe", "Lazarus", "Kwapre",
    "Nzoka", "Jauro", "Birma", "Fwa", "Tumba", "Dlama", "Nuhu", "Zira", "Bitrus",
    "Vandi", "Nggada", "Gimba", "Danjuma"
]

prefixes = ["Alhaji", "Malam", "Doctor", "Pastor", "Chief", "Prince", "Princess", "Rev"]
suffixes = ["Abubakar", "Musa", "Ibrahim", "Aliyu", "Yakubu", "Bitrus", "Danjuma", "Zira", "Vandi", "Nuhu"]
syllables = ["Nga", "Fwa", "Tiz", "Lam", "Bok", "Ngu", "Pwa", "Kiri", "Shaf", "Loru", "Baga", "Dla", "Hoba", "Zar", "Yam", "Kwada'

def make_variants(base_list, prefixes, suffixes, syllables, target=2000, max_attempts=20000):
    items = set(base_list)
    attempts = 0
    while len(items) < target and attempts < max_attempts:
        r = random.random()
        if r < 0.3 and prefixes:
            new = random.choice(prefixes) + " " + random.choice(base_list)
        elif r < 0.6 and suffixes:
            new = random.choice(base_list) + " " + random.choice(suffixes)
        elif r < 0.8 and syllables:
            new = random.choice(syllables) + random.choice(syllables)
        else:
            new = random.choice(base_list) + " " + random.choice(base_list)
        items.add(new)
        attempts += 1

    # Fill with duplicates if still short
    items = list(items)
    while len(items) < target:
        items.append(random.choice(items))
    return items[:target]

random.seed(2025)
all_person_names = make_variants(base_names + person_words_fixed, prefixes, suffixes, syllables, 2000)

# ---------------------------
# STEP 2b. Expand TIME and ANIMAL with variants to 2000
# ---------------------------
time_prefixes = ["Early", "Late", "Mid", "Pre", "Post"]
time_suffixes = ["time", "hour", "day", "night", "season"]
time_syllables = ["Zi", "Sa", "Na", "Ku", "Lo", "Mi", "Ta"]

animal_prefixes = ["Wild", "Big", "Little", "Young", "Old"]
animal_suffixes = ["beast", "cub", "ling", "hunter", "creature"]
animal_syllables = ["Ka", "Mo", "La", "Ti", "Ro", "Zu", "Ba"]

all_time_words = make_variants(time_words, time_prefixes, time_suffixes, time_syllables, 2000)
all_animal_words = make_variants(animal_words, animal_prefixes, animal_suffixes, animal_syllables, 2000)
```

```
# ---------------------------
# STEP 3. Location generator
# ---------------------------
base_places = [
    "Yola", "Jimeta", "Numan", "Ganye", "Gombi", "Hong", "Mubi", "Michika", "Madagali",
    "Maiha", "Fufore", "Song", "Demsa", "Guyuk", "Jada", "Lamurde", "Mayo-Belwa",
    "Shelleng", "Toungo", "Pella", "Uba", "Dirma", "Holma", "Kala'a", "Garkida",
    "Borrong", "Mayo-Lope", "Shuwa", "Mayo-Balewa", "River Benue", "Mayo Ine",
    "Mayo Nguli", "Mayo Sanzu", "Kiri Dam", "Mandara Mountains", "Zumo Hill", "Fali Hills"
]

prefixes_loc = ["New", "Old", "Upper", "Lower", "North", "South", "East", "West", "Mayo", "Wuro", "Gidan", "Bari"]
suffixes_loc = ["Gari", "Ward", "Hill", "Village", "Settlement", "Bridge", "Camp", "Market", "River", "Valley", "Peak", "Forest",
syllables_loc = ["Kwa", "Ngu", "Mayo", "Zar", "Kiri", "Wuro", "Tula", "Nguwa", "Ganye", "Song", "Lam", "Mubi", "Pella", "Hoba", "E

all_places = make_variants(base_places, prefixes_loc, suffixes_loc, syllables_loc, 2000)

# ---------------------------
# STEP 4. Annotation helper
# ---------------------------
def make_annotation(word, label):
    return {
        "data": {"text": word},
        "annotations": [{
            "result": [{
                "value": {
                    "start": 0,
                    "end": len(word),
                    "text": word,
                    "labels": [label]
                },
                "from_name": "label",
                "to_name": "text",
                "type": "labels"
            }]
        }]
    }

# Build datasets
time_tasks = [make_annotation(w, "TIME") for w in all_time_words]          # expanded 2000
animal_tasks = [make_annotation(w, "ANIMAL") for w in all_animal_words]    # expanded 2000
person_tasks = [make_annotation(w, "PERSON") for w in all_person_names]    # expanded 2000
location_tasks = [make_annotation(loc, "LOCATION") for loc in all_places]  # expanded 2000

# ---------------------------
# STEP 5. Merge datasets and save to Drive
# ---------------------------
merged = time_tasks + animal_tasks + person_tasks + location_tasks

# Create the directory in Drive if it doesn't exist
os.makedirs(drive_path, exist_ok=True)

# Save to Drive
drive_file_path = os.path.join(drive_path, "merged_dataset.json")
with open(drive_file_path, "w", encoding="utf-8") as f:
    json.dump(merged, f, indent=2, ensure_ascii=False)

print(f"✅ Saved {len(merged)} tasks to Drive at {drive_file_path}")
print(f"  TIME: {len(time_tasks)}")
print(f"  ANIMAL: {len(animal_tasks)}")
print(f"  PERSON: {len(person_tasks)}")
print(f"  LOCATION: {len(location_tasks)}")
```

```
Your Data is located in /content/data/dataset.conll
✅ Saved 8000 tasks to Drive at /content/drive/MyDrive/NER_nyahoba/notebook/content/data/merged_dataset.json
  TIME: 2000
  ANIMAL: 2000
  PERSON: 2000
  LOCATION: 2000
```

3. Data Annotation

The generated data is then annotated with entity labels and merged into a unified dataset, ensuring sufficient volume and diversity for NER model training while maintaining consistency and quality.

```
import json
import os

# Load your merged dataset
drive_path = '/content/drive/MyDrive/NER_nyahoba/notebook/content/data'
# Define the drive path
merged_dataset_path = os.path.join(drive_path, "merged_dataset.json")

with open(merged_dataset_path, "r", encoding="utf-8") as f:
    data = json.load(f)

conll_lines = []

for task in data:
    text = task["data"]["text"]
    anns = task["annotations"][0]["result"] if task["annotations"] else []

    # Start with "O" for each token
    tokens = text.split()
    labels = ["O"] * len(tokens)

    for ann in anns:
        value = ann["value"]
        start = value["start"]
        end = value["end"]
        label = value["labels"][0]

        # Find which tokens are covered by this annotation
        covered = []
        running_index = 0
        for i, tok in enumerate(tokens):
            token_start = running_index
            token_end = running_index + len(tok)
            if token_end > start and token_start < end:
                covered.append(i)
            running_index = token_end + 1  # +1 for space

        # Assign BIO tags
        for j, idx in enumerate(covered):
            if j == 0:
                labels[idx] = "B-" + label
            else:
                labels[idx] = "I-" + label

    # Append tokens with tags
    for tok, lab in zip(tokens, labels):
        conll_lines.append(f"{tok} {lab}")
    conll_lines.append("")  # Sentence boundary

# Save to file in Drive
drive_conll_path = os.path.join(drive_path, "dataset.conll")
with open(drive_conll_path, "w", encoding="utf-8") as f:
    f.write("\n".join(conll_lines))

print(f"✅ Exported to {drive_conll_path} in CoNLL format")
```

```
✅ Exported to /content/drive/MyDrive/NER_nyahoba/notebook/content/data/dataset.conll in CoNLL format
```

4. Parse CoNLL & Prepare JSONL

This cell parses the CoNLL file (token per line, tag in last column) and saves a JSONL to `/content/prepared/data.jsonl`.

```
from pathlib import Path
import json, os
from collections import Counter

drive_path = '/content/drive/MyDrive/NER_nyahoba/notebook/content' # Define drive_path
conll_path = Path(os.path.join(drive_path, 'data/dataset.conll')) # Use drive_path
if not conll_path.exists():
    raise FileNotFoundError(f'{conll_path} not found. Please run the previous cell to generate it.')

def read_conll(path):
    sentences = []
```

```
        tokens, tags = [], []
        with open(path, 'r', encoding='utf-8') as f:
            for line in f:
                line = line.strip()
                if not line:
                    if tokens:
                        sentences.append((tokens, tags))
                        tokens, tags = [], []
                    continue
                parts = line.split()
                token = parts[0]
                tag = parts[-1] if len(parts) > 1 else 'O'
                tokens.append(token); tags.append(tag)
            if tokens:
                sentences.append((tokens, tags))
        return sentences

    sentences = read_conll(conll_path)
    num_sentences = len(sentences)
    num_tokens = sum(len(s[0]) for s in sentences)
    labels = Counter()
    for toks, tgs in sentences:
        labels.update(tgs)

    print('Sentences:', num_sentences)
    print('Tokens (total):', num_tokens)
    print('Label set:', sorted(labels.keys()))

    prepared_drive_path = os.path.join(drive_path, 'prepared')
    os.makedirs(prepared_drive_path, exist_ok=True)
    jsonl_drive_path = os.path.join(prepared_drive_path, 'data.jsonl')
    with open(jsonl_drive_path, 'w', encoding='utf-8') as outf:
        for toks, tgs in sentences:
            outf.write(json.dumps({'tokens': toks, 'tags': tgs}, ensure_ascii=False) + '\n')
    print(f'Saved prepared JSONL to {jsonl_drive_path}')
```

```
Sentences: 8000
Tokens (total): 16035
Label set: ['B-ANIMAL', 'B-LOCATION', 'B-PERSON', 'B-TIME', 'I-ANIMAL', 'I-LOCATION', 'I-PERSON', 'I-TIME']
Saved prepared JSONL to /content/drive/MyDrive/NER_nyahoba/notebook/content/prepared/data.jsonl
```

Sample annotated sentences

```
# print first 10 samples
import json, itertools, os
drive_path = '/content/drive/MyDrive/NER_nyahoba/notebook/content'
jsonl_drive_path = os.path.join(drive_path, 'prepared', 'data.jsonl') # Use drive_path

with open(jsonl_drive_path, 'r', encoding='utf-8') as f:
    for i, line in enumerate(itertools.islice(f, 10), 1):
        d = json.loads(line)
        print(i, '->', ' '.join([f"{t}/{tg}" for t,tg in zip(d['tokens'], d['tags'])]))
```

```
1 -> KuZi/B-TIME
2 -> Pəshinda/B-TIME Fer/I-TIME pəchingə/I-TIME zekeu/I-TIME
3 -> Pər/B-TIME pəchingə/I-TIME zekeu/I-TIME Pəshinda/I-TIME
4 -> əna/B-TIME əna/I-TIME
5 -> Zəkəu/B-TIME hour/I-TIME
6 -> Hya/B-TIME Sakana/I-TIME
7 -> Pəchi/B-TIME time/I-TIME
8 -> ZiTa/B-TIME
9 -> Mid/B-TIME Pər/I-TIME pəchingə/I-TIME zekeu/I-TIME
10 -> Fer/B-TIME pəchingə/I-TIME zekeu/I-TIME əna/I-TIME
```

5. Baseline: CRF Model

Train a CRF baseline using `sklearn-crfsuite`. This step is fast and useful for benchmarking.

```
# CRF baseline training
import json
from sklearn_crfsuite import CRF, metrics
from sklearn.model_selection import train_test_split
import joblib
import os

drive_path = '/content/drive/MyDrive/NER_nyahoba/notebook/content'
jsonl_drive_path = os.path.join(drive_path, 'prepared', 'data.jsonl')

data = [json.loads(line) for line in open(jsonl_drive_path, encoding='utf-8')]
tokens = [d['tokens'] for d in data]
tags = [d['tags'] for d in data]

def word2features(sent, i):
    word = sent[i]
    features = {
        'bias': 1.0,
        'word.lower()': word.lower(),
        'word.isupper()': word.isupper(),
        'word.istitle()': word.istitle(),
        'word.isdigit()': word.isdigit(),
    }
    if i > 0:
        word1 = sent[i-1]
        features.update({
            '-1:word.lower()': word1.lower(),
            '-1:word.istitle()': word1.istitle(),
            '-1:word.isupper()': word1.isupper(),
        })
    else:
        features['BOS'] = True
    if i < len(sent)-1:
        word1 = sent[i+1]
        features.update({
            '+1:word.lower()': word1.lower(),
            '+1:word.istitle()': word1.istitle(),
            '+1:word.isupper()': word1.isupper(),
        })
    else:
        features['EOS'] = True
    return features

def sent2features(sent):
    return [word2features(sent, i) for i in range(len(sent))]

X = [sent2features(s) for s in tokens]
y = tags
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

crf = CRF(algorithm='lbfgs', c1=0.1, c2=0.1, max_iterations=200)
crf.fit(X_train, y_train)
y_pred = crf.predict(X_test)
labels = [l for l in crf.classes_ if l != 'O']
print(metrics.flat_classification_report(y_test, y_pred, labels=labels, digits=4))

crf_drive_path = os.path.join(drive_path, 'prepared', 'crf_nyahoba.joblib')
joblib.dump(crf, crf_drive_path)
print(f'Saved CRF model to {crf_drive_path}')
```

```
              precision    recall  f1-score   support

      B-TIME     1.0000    0.9953    0.9976       425
      I-TIME     1.0000    1.0000    1.0000       567
    B-PERSON     0.9730    0.9068    0.9387       397
    I-PERSON     1.0000    1.0000    1.0000       351
  B-LOCATION     0.9068    0.9749    0.9396       399
  I-LOCATION     1.0000    1.0000    1.0000       406
    B-ANIMAL     1.0000    0.9974    0.9987       379
    I-ANIMAL     1.0000    1.0000    1.0000       307

    accuracy                         0.9845      3231
   macro avg     0.9850    0.9843    0.9843      3231
weighted avg     0.9852    0.9845    0.9845      3231

Saved CRF model to /content/drive/MyDrive/NER_nyahoba/notebook/content/prepared/crf_nyahoba.joblib
```

6. Transformer Fine-tuning (Hugging Face)

Fine-tune a transformer for token classification. This cell uses Hugging Face `Trainer`. **Requires GPU** for reasonable speed.

```
# Rerun the transformer fine-tuning template
# (This may be slow on Colab free tier and requires GPU for reasonable speed)

# Ensure you have a GPU runtime enabled:
# Go to Runtime -> Change runtime type -> GPU

from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForTokenClassification, TrainingArguments, Trainer, DataCollatorForTokenClassifica
import json, os

drive_path = '/content/drive/MyDrive/NER_nyahoba/notebook/content'
jsonl_drive_path = os.path.join(drive_path, 'prepared', 'data.jsonl')

MODEL_NAME = "xlm-roberta-base"  # change to an Afro model if you prefer (e.g., Davlan/afro-xlmr-base)

data = [json.loads(line) for line in open(jsonl_drive_path, encoding='utf-8')]
dataset = Dataset.from_list([{'tokens': d['tokens'], 'tags': d['tags']} for d in data])

unique_labels = sorted({lab for d in data for lab in d['tags']})
label2id = {l:i for i,l in enumerate(unique_labels)}
id2label = {i:l for l,i in label2id.items()}

print('Labels:', unique_labels)
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, use_fast=True)

def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(examples['tokens'], is_split_into_words=True, truncation=True, padding='max_length', max_length=12
    labels = []
    for i, label in enumerate(examples['tags']):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        previous_word_idx = None
        label_ids = []
        for word_idx in word_ids:
            if word_idx is None:
                label_ids.append(-100)
            elif word_idx != previous_word_idx:
                label_ids.append(label2id[label[word_idx]])
            else:
                lab = label[word_idx]
                if lab.startswith('B-'):
                    lab = 'I-' + lab.split('-',1)[1]
                label_ids.append(label2id.get(lab, label2id[label[word_idx]]))
            previous_word_idx = word_idx
        labels.append(label_ids)
    tokenized_inputs['labels'] = labels
    return tokenized_inputs

tokenized = dataset.map(tokenize_and_align_labels, batched=True)
tokenized = tokenized.train_test_split(test_size=0.1)
model = AutoModelForTokenClassification.from_pretrained(MODEL_NAME, num_labels=len(unique_labels), id2label=id2label, label2id=labe

args = TrainingArguments(
    output_dir='/content/nyahoba_ner_output', # Keep local output for trainer
    eval_strategy='epoch',
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    save_total_limit=2,
    logging_steps=50,
)

data_collator = DataCollatorForTokenClassification(tokenizer)

trainer = Trainer(
    model,
    args,
    train_dataset=tokenized['train'],
    eval_dataset=tokenized['test'],
    data_collator=data_collator,
    tokenizer=tokenizer,
)

print('Starting transformer training (this may take a long time).')
trainer.train()

# Save the model to Drive
# Save the model to Drive
transformer_drive_path = os.path.join(drive_path, 'prepared', 'nyahoba-ner-model')
trainer.save_model(transformer_drive_path)
print(f'Saved transformer model to {transformer_drive_path}')
```

```
Labels: ['B-ANIMAL', 'B-LOCATION', 'B-PERSON', 'B-TIME', 'I-ANIMAL', 'I-LOCATION', 'I-PERSON', 'I-TIME']
/usr/local/lib/python3.12/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
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: 100%                                   25.0/25.0 [00:00<00:00, 2.63kB/s]

config.json: 100%                                   615/615 [00:00<00:00, 68.9kB/s]

sentencepiece.bpe.model: 100%                                   5.07M/5.07M [00:00<00:00, 10.3MB/s]

tokenizer.json: 100%                                   9.10M/9.10M [00:00<00:00, 15.9MB/s]

Map: 100%                                   8000/8000 [00:01<00:00, 5454.20 examples/s]

model.safetensors: 100%                                   1.12G/1.12G [00:15<00:00, 104MB/s]
Some weights of XLMRobertaForTokenClassification were not initialized from the model checkpoint at xlm-roberta-base and are newly i
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/tmp/ipython-input-2122161499.py:66: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__i
  trainer = Trainer(
Starting transformer training (this may take a long time).
/usr/local/lib/python3.12/dist-packages/notebook/notebookapp.py:191: SyntaxWarning: invalid escape sequence '\/'
  | |_| | '_ \/ _` / _` |  _/ -_)
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: chahyaandida (chahyaandida-modibbo-adama-university) to https://api.wandb.ai. Use `wandb login --rel
creating run (0.0s)
Tracking run with wandb version 0.21.3
Run data is saved locally in /content/wandb/run-20250920_175938-mfsphcrb
Syncing run wise-brook-3 to Weights & Biases (docs)
View project at https://wandb.ai/chahyaandida-modibbo-adama-university/huggingface
View run at https://wandb.ai/chahyaandida-modibbo-adama-university/huggingface/runs/mfsphcrb
[2700/2700 17:23, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss
1	0.139900	0.037692
2	0.036900	0.028062
3	0.020200	0.028856

```
Saved transformer model to /content/drive/MyDrive/NER_nyahoba/notebook/content/prepared/nyahoba-ner-model
```

7. Inference Examples

Load saved models and run inference on sample texts. Edit the sample sentences as needed.

```
# CRF inference
import joblib, json
from pathlib import Path
import os

drive_path = '/content/drive/MyDrive/NER_nyahoba/notebook/content'
crf_drive_path = Path(os.path.join(drive_path, 'prepared', 'crf_nyahoba.joblib'))

if crf_drive_path.exists():
    crf = joblib.load(crf_drive_path)
    sample = ['Zəkəu', 'Sakana', 'əna', 'Hong', 'Gombi', 'pəshinda', 'kwa', 'thla']
    def sent2features(sent):
        def word2features(sent, i):
            word = sent[i]
            features = {
                'bias': 1.0,
                'word.lower()': word.lower(),
                'word.isupper()': word.isupper(),
                'word.istitle()': word.istitle(),
                'word.isdigit()': word.isdigit(),
            }
            if i > 0:
                word1 = sent[i-1]
                features.update({
                    '-1:word.lower()': word1.lower(),
                    '-1:word.istitle()': word1.istitle(),
                    '-1:word.isupper()': word1.isupper(),
                })
            else:
                features['BOS'] = True
            if i < len(sent)-1:
                word1 = sent[i+1]
                features.update({
                    '+1:word.lower()': word1.lower(),
                    '+1:word.istitle()': word1.istitle(),
                    '+1:word.isupper()': word1.isupper(),
                })
            else:
                features['EOS'] = True
            return features
        return [word2features(sent, i) for i in range(len(sent))]
    print('CRF prediction:', crf.predict([sent2features(sample)]))
else:
    print('CRF model not found at', crf_drive_path)

# Transformer inference (if model exists)
from transformers import AutoTokenizer, AutoModelForTokenClassification, pipeline
transformer_drive_path = Path(os.path.join(drive_path, 'prepared', 'nyahoba-ner-model'))

if transformer_drive_path.exists():
    tokenizer = AutoTokenizer.from_pretrained(transformer_drive_path, use_fast=True)
    model = AutoModelForTokenClassification.from_pretrained(transformer_drive_path)
    nlp = pipeline('token-classification', model=model, tokenizer=tokenizer, aggregation_strategy='simple')
    print(nlp('chiwar'))
else:
    print('Transformer model not found at', transformer_drive_path)
```

```
CRF prediction: [['B-TIME' 'I-TIME' 'I-TIME' 'B-LOCATION' 'I-LOCATION' 'I-LOCATION'
  'B-ANIMAL' 'I-ANIMAL']]
Device set to use cuda:0
[{'entity_group': 'ANIMAL', 'score': np.float32(0.99993235), 'word': 'chiwar', 'start': 0, 'end': 6}]
```

8. Save & Export

- Use `File -> Download .ipynb` in Colab to download the notebook.
- Download trained artifacts from `/content/prepared` (you can zip and download them in Colab).