**NAME ENTITY RECOGNITION (NER) USING BERT BASE MODEL**

**(TRAINING)**

**Table of Contents**

[**Introduction 3**](#_heading=)

[BERT Model 3](#_heading=h.gjdgxs)

[HuggingFace Hub 4](#_heading=h.30j0zll)

[**Requirements 5**](#_heading=h.3znysh7)

[**Script Overview 5**](#_heading=h.tyjcwt)

[1. Argument Parsing 5](#_heading=h.3dy6vkm)

[2. Configuration Loading 6](#_heading=h.1t3h5sf)

[3. Tokenization and Label Alignment 6](#_heading=h.2s8eyo1)

[4. Main Function 7](#_heading=h.3rdcrjn)

[**Running the Script 10**](#_heading=h.lnxbz9)

[Code Location 10](#_heading=h.35nkun2)

[Post Login and navigation to the directory 10](#_heading=h.1ksv4uv)

[Step 2: Navigate to the Configuration File 11](#_heading=h.44sinio)

[Step 3: Adjust the Parameters in config.json 11](#_heading=h.2jxsxqh)

[Step 4: Verify Logins for Hugging Face and Weights & Biases (W&B) 12](#_heading=h.3j2qqm3)

[Step 4.1: Login to Hugging Face 12](#_heading=h.1y810tw)

[Step 4.2: Login to Weights & Biases 12](#_heading=h.4i7ojhp)

[Step 4.3: Verify Login 12](#_heading=h.2xcytpi)

[Step 5: Run the Fine-Tuning Program 12](#_heading=h.1ci93xb)

[Explanation of the json configuration 13](#_heading=h.2bn6wsx)

# Introduction

**References:** [dslim/bert-base-NER · Hugging Face](https://huggingface.co/dslim/bert-base-NER)

## BERT Model

BERT-BASE-NER model is a fine-tuned version of BERT, specifically designed for Named Entity Recognition (NER). This model excels at identifying and categorizing different types of entities within a text, making it highly effective for tasks involving text analysis and information extraction. It can recognize and classify four main types of entities:

**Location (LOC)**: Identifies geographical places, such as cities, countries, and landmarks.

**Organization (ORG)**: Detects entities like companies, institutions, and government bodies.

**Person (PER)**: Recognizes names of individuals.

**Miscellaneous (MISC)**: Captures other entities that don’t fit into the above categories, such as events, works of art, and nationalities.

As in the dataset, each token will be classified as one of the following classes:

| **Abbreviation** | **Description** |
| --- | --- |
| I-LOC | Location |
| B-LOC | Beginning of a location right after another location |
| I-ORG | Organization |
| B-ORG | Beginning of an organization right after another organization |
| I-PER | Person’s name |
| B-PER | Beginning of a person’s name right after another person’s name |
| I-MISC | Miscellaneous entity |
| B-MISC | Beginning of a miscellaneous entity right after another miscellaneous entity |
| O | Others |

**Code location:** /home/ai001/INT4TB/modeltuning/finetuning\_nermodel/

## HuggingFace Hub

HuggingFace is an open platform for Machine learning which provides access to a wide range of pre-trained open-source models, datasets and other resources.

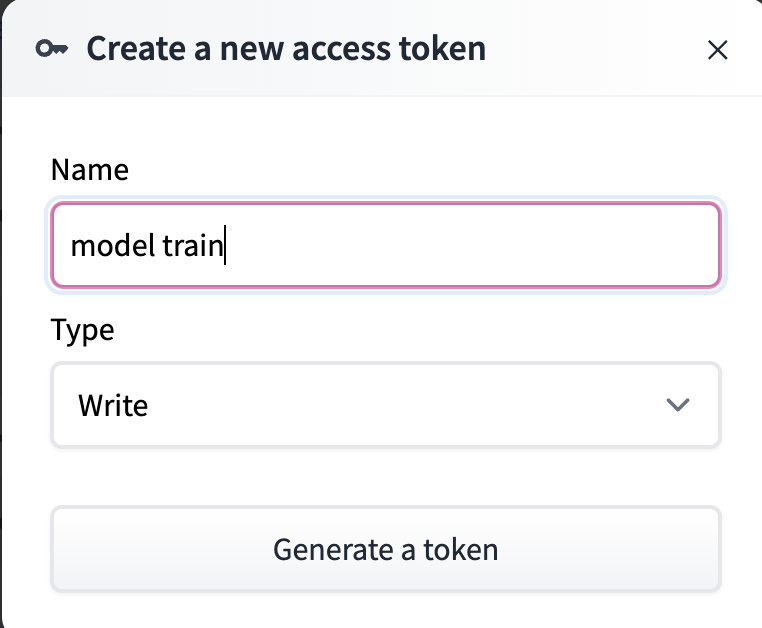
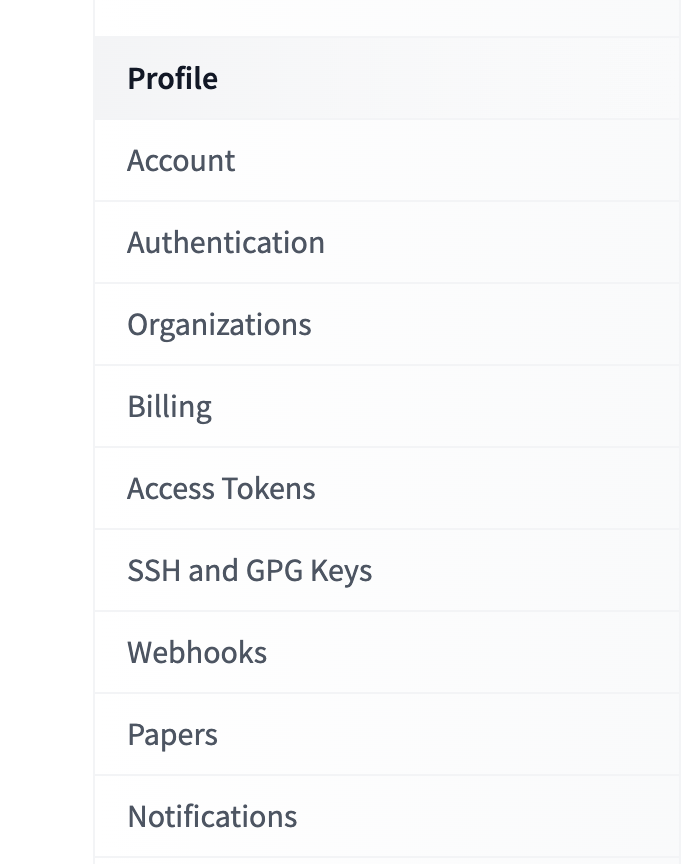
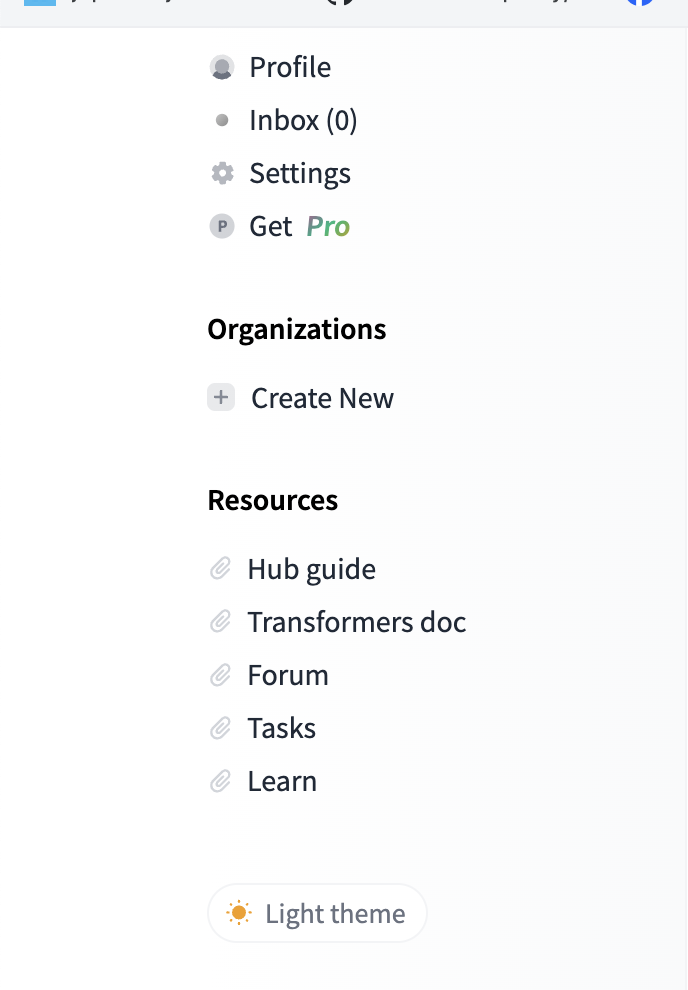
We use this platform to store our project datasets, trained models and the models under experimentation.

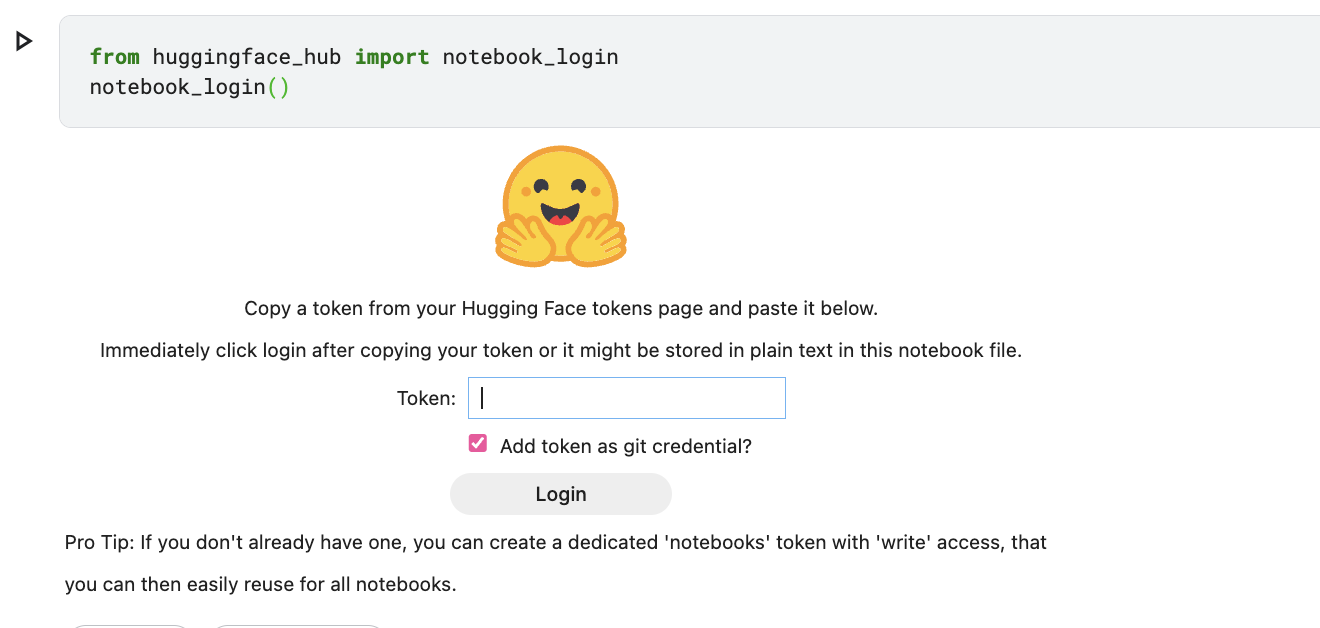
Before starting, ensure you have the secret token of hugging face.

The secret token can be created by : Settings -> Access Tokens -> New Token

Here you can generate a new token with Type - Write

The processes are shown in images below:





## 

# Requirements

* Python 3.6+
* Install the necessary Python packages:

pip install transformers datasets wandb huggingface\_hub

import argparse

import json, os

import wandb

from huggingface\_hub import login

from transformers import (AutoModelForTokenClassification, AutoTokenizer, Trainer, TrainingArguments, DataCollatorForTokenClassification)

from datasets import load\_dataset

# 

# Script Overview

The script is designed to fine-tune a pre-trained transformer model for NER tasks using a custom configuration file. The main components of the script include argument parsing, configuration loading, dataset tokenization, model training, and model uploading to the Hugging Face Hub

### 1. Argument Parsing

The script begins by parsing command-line arguments to obtain the path to the configuration file.

**argparse.ArgumentParser**: Initializes an argument parser object that processes command-line arguments.

**add\_argument method**: Adds a command-line argument (--config) which specifies the path to the configuration file. This argument is required for the script to run.

**parse\_args method**: Parses the command-line arguments and returns them as an object. This allows the script to access the provided configuration file path.

def parse\_args():

parser = argparse.ArgumentParser(description="Fine-tune a NER model.")

parser.add\_argument('--config', type=str, required=True, help="Path to the configuration file.")

return parser.parse\_args()

### 2. Configuration Loading

This part of the script loads the configuration parameters from a JSON file.

**Open the JSON file**: The script reads the configuration file specified by the user.

**Load the JSON content**: The JSON content is parsed and loaded into a Python dictionary. This dictionary contains all the configuration parameters needed for the script, such as model checkpoints, dataset names, and hyperparameters.

def load\_config(config\_path):

with open(config\_path) as f:

config = json.load(f)

return config

### 

### 3. Tokenization and Label Alignment

This function tokenizes the input texts and aligns the NER labels with the tokenized words. Proper label alignment is crucial for token classification tasks.

**Tokenize inputs**: The tokenizer processes the input text tokens, adding padding and truncation as necessary.

**Align labels**: Labels are aligned with the tokenized words. Special handling ensures that labels correspond correctly to their respective tokens, accounting for cases where words are split into multiple tokens.

def tokenize\_and\_align\_labels(examples, tokenizer, label\_all\_tokens=True):

tokenized\_inputs = tokenizer(examples["tokens"], truncation=True, padding='max\_length', is\_split\_into\_words=True)

labels = []

for i, label in enumerate(examples["ner\_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(label[word\_idx])

else:

label\_ids.append(-100)

previous\_word\_idx = word\_idx

labels.append(label\_ids)

tokenized\_inputs["labels"] = labels

return tokenized\_inputs

### 

### 4. Main Function

The main function orchestrates the entire process, including logging in, loading datasets, tokenizing data, training the model, and saving/uploading the trained model.

* **Logging In:** The script logs into WandB and the Hugging Face Hub using tokens provided in the configuration file.

try:

login(token=config["huggingface\_hub"])

except Exception as e:

print("error logging hub", str(e))

wandb.login()

wandb.init(project=config["wandb\_project"])

**Hugging Face Hub Login**: Authenticates the user with the Hugging Face Hub using a provided API token. Refer page no. 3 and 4 for token creation.

**WandB Login**: Authenticates the user with WandB to enable experiment tracking and visualization.

**Initialize WandB Project**: Starts a new WandB run for the specified project.

* **Loading the Dataset:** The dataset specified in the configuration file is loaded. It Uses the Hugging Face datasets library to load a dataset by name. This dataset includes the text and associated NER labels needed for training.

dataset = load\_dataset(config["dataset\_name"])

print(dataset)

* **Tokenizer and Model Initialization:** The tokenizer and model are loaded from the pre-trained model checkpoint specified in the configuration.

tokenizer = AutoTokenizer.from\_pretrained(config["model\_checkpoint"])

model = AutoModelForTokenClassification.from\_pretrained(

config["model\_checkpoint"],

num\_labels=len(dataset["train"].features["ner\_tags"].feature.names)

)

**Load tokenizer**: Initializes a tokenizer from the specified pre-trained model checkpoint. This tokenizer will preprocess the input text.

**Load model**: Initializes a token classification model from the pre-trained model checkpoint, setting the number of output labels to match the dataset's NER tags.

* **Tokenizing the Dataset:** The dataset is tokenized using the tokenize\_and\_align\_labels function.

tokenized\_datasets = dataset.map(lambda x: tokenize\_and\_align\_labels(x, tokenizer), batched=True)

Applies the tokenization and label alignment function to the entire dataset.

This prepares the data for training.

* **Training Arguments:** Training arguments are set up according to the configuration file. These parameters control the training process as they Specify various training parameters such as output directory, evaluation strategy, learning rate, batch sizes, number of epochs, weight decay, and reporting mechanisms.

training\_args = TrainingArguments(

output\_dir=config["output\_dir"],

evaluation\_strategy=config["evaluation\_strategy"],

learning\_rate=config["learning\_rate"],

per\_device\_train\_batch\_size=config["train\_batch\_size"],

per\_device\_eval\_batch\_size=config["eval\_batch\_size"],

num\_train\_epochs=config["num\_train\_epochs"],

weight\_decay=config["weight\_decay"],

report\_to=config["report\_to"]

)

Once the Trainer is initialized with the model, training arguments, data collator, and datasets, then the training process begins.

* **Data Collator**: Ensures correct formatting and batching of data during training.

data\_collator = DataCollatorForTokenClassification(tokenizer)

* **Initialize Trainer and Training**: Combines the model, training arguments, data collator, training dataset, and evaluation dataset into a training loop, then the training process starts.

trainer = Trainer(

model=model,

args=training\_args,

data\_collator=data\_collator,

train\_dataset=tokenized\_datasets["train"],

eval\_dataset=tokenized\_datasets["validation"],

)

trainer.train()

* **Saving and Uploading the Model:** After training, the model and tokenizer are saved locally and pushed to the Hugging Face Hub.

model.save\_pretrained(config["model\_save\_path"])

tokenizer.save\_pretrained(config["model\_save\_path"])

try:

model.push\_to\_hub(config["hub\_model\_name"])

tokenizer.push\_to\_hub(config["hub\_model\_name"])

except Exception as e:

print("Error pushing to hub:", str(e))

* **Finishing the WandB Run:** Finally, the WandB run is finished. The code below ends the WandB run, finalizing the experiment tracking.

wandb.finish()

# 

# Running the Script

Here is the refined document with detailed steps for running the fine-tuning process on a server:

## Code Location

1. **Login to the server**:

ssh -p 39049 ai001@213.34.245.81

Enter password

1. **Navigate to the code directory**:

cd /home/ai001/INT4TB/modeltuning/finetuning\_nermodel/

## Post Login and navigation to the directory

After logging in the server the steps to follow are:

**Step 1: Activate the Environment**

Activate the virtual environment:

source env/bin/activate

### Step 2: Navigate to the Configuration File

Go to the path:

cd /home/ai001/INT4TB/modeltuning/finetuning\_nermodel/config.json

### Step 3: Adjust the Parameters in config.json

Modify the config.json configuration file as needed. Example is given below:

{

"wandb\_project": "ner\_project\_v6",

"dataset\_name": "procit001/conll2003susantest",

"model\_checkpoint": "bert-base-cased",

"output\_dir": "./results",

"evaluation\_strategy": "epoch",

"learning\_rate": 2e-5,

"train\_batch\_size": 8,

"eval\_batch\_size": 8,

"num\_train\_epochs": 5,

"weight\_decay": 0.01,

"save\_steps":1000,

"report\_to": "wandb",

"model\_save\_path": "./ner\_model\_v1",

"huggingface\_hub":"hf\_CDgAivSuIKePjBfUEBXtuszjKqtQyWTaYc",

"wandb":"allow",

"push\_to\_hub":"True",

"load\_best\_model\_at\_end": "True",

"hub\_model\_name": "procit001/ner\_model\_4",

"cache\_dir":"/home/ai001/INT4TB/modeltuning/finetuning\_nermodel/cache/"

}

### 

### Step 4: Verify Logins for Hugging Face and Weights & Biases (W&B)

#### Step 4.1: Login to Hugging Face

Log in using the terminal and enter your token:

huggingface-cli login

login()

**Token:** Your Hugging Face token with write access to post the model to Hugging Face.

**Note:** The steps are provided in HuggingFace Hub in Page no. 3 and 4

#### Step 4.2: Login to Weights & Biases

Log in to W&B:

wandb login

**Token**: “your-wandb-token”

#### Step 4.3: Verify Login

To verify your Hugging Face login:

huggingface-cli whoami

### Step 5: Run the Fine-Tuning Program

Run the fine-tuning process. Keep the terminal on until the process completes.

python main.py --config path/to/config.json

For server:

python main.py --config config.json

## 

## Explanation of the json configuration

{

"wandb\_project": "ner\_project\_v6",

"dataset\_name": "procit001/conll2003susantest",

"model\_checkpoint": "bert-base-cased",

"output\_dir": "./results",

"evaluation\_strategy": "epoch",

"learning\_rate": 2e-5,

"train\_batch\_size": 8,

"eval\_batch\_size": 8,

"num\_train\_epochs": 5,

"weight\_decay": 0.01,

"save\_steps":1000,

"report\_to": "wandb",

"model\_save\_path": "./ner\_model\_v1",

"huggingface\_hub":"hf\_CDgAivSuIKePjBfUEBXtuszjKqtQyWTaYc",

"wandb":"allow",

"push\_to\_hub":"True",

"load\_best\_model\_at\_end": "True",

"hub\_model\_name": "procit001/ner\_model\_4",

"cache\_dir":"/home/ai001/INT4TB/modeltuning/finetuning\_nermodel/cache/"

}

The above JSON configuration is for setting up and training a Named Entity Recognition (NER) model using a specific dataset and logging the training process with Weights & Biases (wandb). Here’s an explanation of each field:

**wandb\_project**: "ner\_project\_v6" - The name of the Weights & Biases project where training logs and metrics will be stored.

**dataset\_name**: "procit001/conll2003susantest" - The name of the dataset used for training and evaluation. In this case, it appears to be a custom version of the CoNLL-2003 dataset.

**model\_checkpoint**: "bert-base-cased" - The pre-trained BERT model (cased version) used as the starting point for fine-tuning.

**output\_dir**: "./results" - The directory where the model's output (e.g., checkpoints, logs) will be saved.

**evaluation\_strategy**: "epoch" - The frequency of evaluation during training. Here, evaluation is performed at the end of each epoch.

**learning\_rate**: 2e-5 - The learning rate used for training the model.

**train\_batch\_size**: 8 - The number of samples per batch used during training.

**eval\_batch\_size**: 8 - The number of samples per batch used during evaluation.

**num\_train\_epochs**: 5 - The number of epochs (complete passes through the training dataset) to train the model.

**weight\_decay**: 0.01 - The weight decay (L2 regularization) rate applied to prevent overfitting.

**report\_to**: "wandb" - Indicates that training metrics and logs will be reported to Weights & Biases.

**model\_save\_path**: "./ner\_model" - The directory where the final trained model will be saved.

**huggingface\_hub**: "your-hugging-face-token" - The token for authenticating and pushing the model to the Hugging Face Hub.

**wandb**: "allow" - Enables the integration with Weights & Biases for tracking the training process.

**hub\_model\_name**: "procit002/ner\_model\_4\_updated\_by\_susan" - The name under which the trained model will be saved on the Hugging Face Hub.

**cache\_dir**: "/home/ai001/INT4TB/modeltuning/finetuning\_nermodel/cache/" - Specifies the directory where datasets and other temporary files are cached. This helps in reusing downloaded data and avoiding redundant downloads, which can save time and bandwidth. It is particularly useful when working with large datasets or when internet connectivity is intermittent.

**push\_to\_hub**:"True" - Indicates whether the model should be pushed to the Hugging Face Hub after training.

**load\_best\_model\_at\_end**: "True" - Indicates whether to load the best model found during training at the end of the training process. When set to True, the script will keep track of the best-performing model during training (based on evaluation metrics) and reload it at the end. This ensures that the final saved model is the best version encountered during training.

**save\_steps**: 1000 - Specifies how often (in steps) the model should be saved during training. This helps in creating checkpoints during training, so you can resume training from the last saved state in case of interruptions. It also provides intermediate versions of the model that can be analyzed or used if needed.