National College of Ireland

Configuration Manual

MSc Research Project
MSc in Artificial Intelligence

Animesh Kumar Rai
Student ID: x23194545

School of Computing
National College of Ireland

Supervisor: Dr. Muslim Jameel Syed

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Animesh Kumar Rai
Student ID:	x23194545
Programme:	MSc in Artificial Intelligence
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr. Muslim Jameel Syed
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	1053
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Animesh Kumar Rai
Date:	29th January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Animesh Kumar Rai
x23194545

1 Introduction

The manual has all the important information on system, hardware and software specifications. It also outlines the process for execution of the study done on the **"Safeguarding Sensitive Data - Detection in Unstructured Text Using Cutting-Edge Transformer Architectures"**. Section 2 tells about the system specifications like hardware and software setups which are utlised for the process of research. Section 3 covers how to set up the environment, import libraries, and do pre-processing. The fourth section discusses model development and assessment.

2 System Configuration

This section contains the hardware and software requirements necessary for running BERT-based models, including namely DistilBERT,, RoBERTa, DeBERTa and, Longformer, for our research. Tthe project is implemented on Google Colab utilizing the a T4 GPU setup, which provides the compuutational power required for training and the inference model.

2.1 Hardware Requirements

Table 1: Hardware Requirements

Component	**Specification**
Operating System	Any Operating System
Processor	Minimum Intel® Core™ i5
Graphics	NVIDIA® Tesla T4 GPU (Google Colaboratory)
GPU Compute	Supports CUDA for TensorFlow and PyTorch
Processor Speed	3.2GHz

The hardware setup provided in table 1 ensures smooth execution of transformer-based models by leveraging Google Colaboratory's T4 GPU for computationally intensive operations.

2.2 Software Requirements

The software setup in table 2 allowed efficient training and evaluation of the models. Utilizing Google Colaboratory's T4 GPU ensured the project's computational needs were met effectively while maintaining compatibility with essential frameworks and libraries.

Table 2: Software Requirements

Component	**Specification**
Web Browser	Google Chrome
Programming Language	Python (TensorFlow & PyTorch support)
Development Platform	Google Colaboratory (T4 GPU enabled)
Version Control	GitHub
Libraries and Tools	Transformers (Hugging Face), NumPy, Pandas, Matplotlib, Torch, TensorFlow

This setup ensures the smooth execution of tasks such as data preprocessing, modelling, and inference i.e, extracting result while thee addressing the computational needs of transformer-based models.

Other Dependencies: Stable Internet connection for upload the dataset and fine-tuned model on google drive, so to run the notebook on google collaboratory.

3 Setting Environment

For the implementtaion of my project, I have utilised Google colaboratory which is the really great tool for Python-based works such as data analysis tasks and machine learning modellings on cloud. I used T4 GPU enabled environment of google colaboratory environment for the model building and model inferencing. jupyter files can be run through this.

1. First go to google collab page : with this link [1]

2. we have to sign-in google account to use this laboratory. Page will look like this 1.

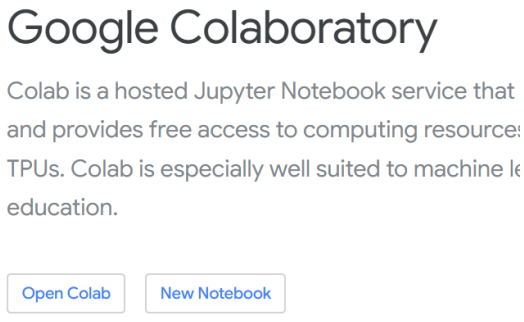

Figure 1: Google Colaboratory

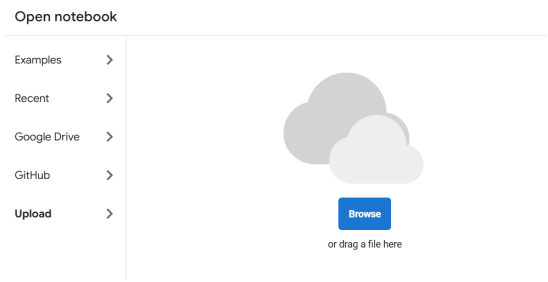

Figure 2: Collab Upload files

3. Then select open collab and then you get option to upload the files and folder 2. browse it and upload notebooks (.ipynb files) provided.

   Alternative: As shown in figure 3 you can click on github option and upload .ipynb notebook from my publicly available github repository . [2]

[1] `https://colab.google/`
[2] `https://github.com/animesh-rai/x23194545_Sensitive_data_detection/tree/main`

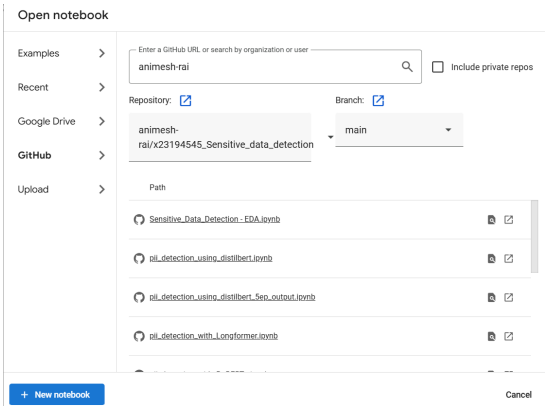

Figure 3: Open notebook through GitHub option

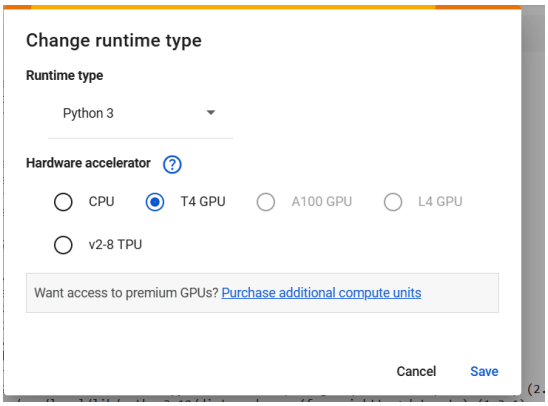

Figure 4: Selecting T4 GPU for acceleration

4. After notebook gets opened, before running the each shell for execution, make sure to enable T4 GPU setup by selecting T4 option from right side connect drop down and then save it and click connect option to notebook will run with GPU configuration.

5. Fined-tuned models for sensitive data detection project is uploaded in my shared google drive [3] which can be downloaded and upload in folder named 'PII_models' under MyDrive of google account where google collab notebook is running, so it will be easier to load the model without any hussel for inferencing with notebook named "sensitive_data_inference_using_all_models.ipynb"

4 Implementation Steps

4.1 Data Load: Learning Agency Lab - PII Data Detection

The project relies on a dataset available on Kaggle [4] which comprising close to 22,000 essays created by students engaging in a massively open online course. User can easily download the dataset from kaggle website or use it from available google drive [5]. first upload the file in 'Dataset' folder under your drive and then run the below command.

```
import datasets
# Load dataset
dataset = datasets.load_dataset('json', data_files='/content/drive/MyDrive/Dataset/pii-detection-removal-from-educational-data/train.json')
# Convert to DataFrame and preprocess
df = dataset['train'].to_pandas()
```

Figure 5: Dataset Load from Drive

[3] https://drive.google.com/drive/folders/1OhDmoqP-g6Xn2DqGkGqSx2zTErDQa0Og?usp=sharing

[4] https://www.kaggle.com/competitions/pii-detection-removal-from-educational-data/data

[5] https://drive.google.com/drive/folders/112Zw_ySDtL0t5dQ-TuZfgr6WSK4vRGOT?usp=drive_link

4.2 Importing Libraries

Importing libraries (packages) is easier since its google collab notebook and all the import and pip statement already integrated with the code blocks. snippet of code blocks are given below.

```
#Import packages
import torch
from torch.utils.data import DataLoader
from transformers import AutoTokenizer, AutoModelForTokenClassification, TrainingArguments, Trainer, DataCollatorForTokenCla
from sklearn.metrics import classification_report, precision_recall_fscore_support
from collections import Counter
import datasets
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
import pandas as pd
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import gc
import re
import random
from itertools import chain
```

Figure 6: Libraries Used Development

4.3 Exploratory Data Analysis

"Sensitive_Data_Detection - EDA.ipynb" jupyter notebook contains functions and experimentations for data exploration step. below is code snippet in figure 7

```
plot_label_distribution(df)
# Print the percentage of "O" labels
total_labels = sum(label_counts.values())
o_percentage = (label_counts['O'] / total_labels) * 100
print(f"'O' label (in percentage): {o_percentage:.2f}")
```

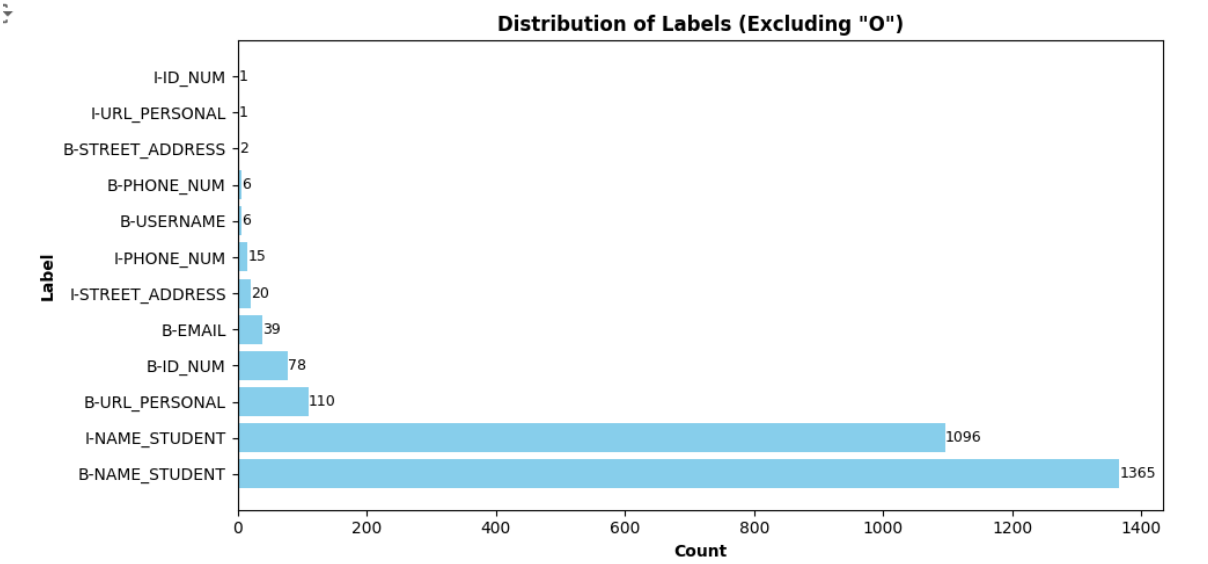

Figure 7: EDA On Sensitive detection Dataset

4.4 Dataset Split for Training and Testing

Each jupyter notebook contains step to split the dataset into training and testing for modelling. below is code snippet in figure 8

```
# Split Dataset
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
train_dataset = datasets.Dataset.from_pandas(train_df)
test_dataset = datasets.Dataset.from_pandas(test_df)
dataset_dict = datasets.DatasetDict({'train': train_dataset, 'test': test_dataset})
```

Figure 8: Train-Test Split

4.5 Tokenization and Label Alignment

Below code in figure 9 taken from 'pii_detector_using_deberta_b.ipynb' file which converts input tokens and their associated labels into a format suitable for the deBERTa modelling. Similarily, all the other models such as roberta, longformer and distilbert have this step too.

```
#Define function to tokenize and align labels
def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(
        examples["tokens"],
        is_split_into_words=True,
        truncation=True,
        padding='max_length',
        max_length=1024,
        return_offsets_mapping=True
    )

    batch_original_tokens = []
    batch_tokenized_tokens = []
    batch_label_ids = []
    batch_input_ids = []
    batch_attention_masks = []
    batch_token_type_ids = []

    for i, label in enumerate(examples["labels"]):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        original_tokens = examples["tokens"][i]
        tokenized_tokens = tokenizer.convert_ids_to_tokens(tokenized_inputs["input_ids"][i])

        previous_word_idx = None
        label_ids = []
        original_token_list = []
        tokenized_token_list = []
```

Figure 9: Tokenize and Align

4.6 Initialize Tokenizer and Model

Below code in figure 10 taken from 'pii_detector_using_deberta_b.ipynb' file which initialize tokens and model for the deBERTa modelling. Similarily, all the other models such as roberta, longformer and distilbert have this step too.

```
# Tokenizer and Model
tokenizer = AutoTokenizer.from_pretrained("microsoft/deberta-v3-base")
model = AutoModelForTokenClassification.from_pretrained("microsoft/deberta-v3-base", num_labels=num_labels, id2label=id2labe
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
model.to(device)
```

Figure 10: Model and Token Initialization

4.7 Handle Label Imbalance Script

Below code in figure 11 taken from 'pii_detector_using_deberta_b.ipynb' file which is build to computes weights for each class to handle label imbalance during training.

```
# Calculate class weights
class_weights = compute_class_weight('balanced', classes=np.unique(all_labels), y=all_labels)
class_weights = torch.tensor(class_weights, dtype=torch.float).to(device)
```

Figure 11: Handle Label Imbalance function

4.8 Custom Compute Metrics Script

Below code in figure 12 taken from 'pii_detector_using_deberta_b.ipynb' file which is custom build to computes metrics for model performance.

```
In [ ]:
def compute_metrics(p):
    predictions, labels = p
    predictions = np.argmax(predictions, axis=2)

    true_labels = [[id2label[label] for label in doc if label != -100] for doc in labels]
    true_predictions = [
        [id2label[pred] for pred, label in zip(doc, labels[i]) if label != -100]
        for i, doc in enumerate(predictions)
    ]

    # Flatten the lists
    true_labels_flat = [item for sublist in true_labels for item in sublist]
    true_predictions_flat = [item for sublist in true_predictions for item in sublist]

    # Calculate metrics
    results = precision_recall_fscore_support(true_labels_flat, true_predictions_flat, average='weighted')

    # Compute classification report
    class_report = classification_report(
        true_labels_flat, true_predictions_flat, labels=all_labels, zero_division=0  # Explicitly set labels and handle zer
    )

    print("Classification Report:\n", class_report)
    return {
        "precision": results[0],
        "recall": results[1],
        "f1": results[2],
        "accuracy": (results[2] * results[1])
    }
```

Figure 12: Custom Compute Metrics Function

4.9 Training Argument Function

Below code in figure 13 taken from 'pii_detector_using_deberta.ipynb' file which is custom build to computes metrics for model performance. Similarily other models codes are there in their respective ipynb file.

```python
# Define training arguments with optimizations
training_args = TrainingArguments(
    output_dir='./results',
    eval_strategy="epoch",              # Use eval_strategy to avoid deprecation warning
    save_strategy="epoch",              # Save model at each epoch
    load_best_model_at_end=True,        # Load the best model after training
    metric_for_best_model="loss",       # Use loss to identify the best model
    greater_is_better=False,            # Lower loss is better
    learning_rate=1e-5,                 # Reduced learning rate for smoother convergence
    per_device_train_batch_size=2,      # Adjust batch size as needed
    per_device_eval_batch_size=2,
    gradient_accumulation_steps=4,      # Simulate larger batch size
    num_train_epochs=5,                 # Set the desired number of epochs
    weight_decay=0.01,                  # Regularization to avoid overfitting
    fp16=True,                          # Mixed precision for memory optimization
    max_grad_norm=1.0,                  # Gradient clipping to stabilize training
    logging_strategy="steps",           # Log at each step
    logging_dir='./logs',                  # Directory for logging (optional)
    logging_steps=10,                   # Adjust logging frequency as needed
    report_to='none',                      # Disable external logging tools
)

# Add an early stopping callback
early_stopping_callback = EarlyStoppingCallback(early_stopping_patience=2)

# Initialize the trainer with the callback
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets['train'],
    eval_dataset=tokenized_datasets['test'],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    callbacks=[early_stopping_callback, loss_logger]  # Include early stopping and loss logger
)
```

Figure 13: Training Argument Script for DeBERTa modelling

4.10 Model Training and Evaluation

Below code in figure 14 taken from 'pii_detector_using_deberta.ipynb' file which is used to train and evaluate model performance. Similarily, other models codes are there in their respective ipynb file which is used for fine tuning models for sensitive data detection and the result of function are in figure 15 and figure 16.

In [ ]:

```python
# Train the model
trainer.train()

# Evaluate the model
eval_results = trainer.evaluate()
print(eval_results)


# Plot the loss per epoch
plt.figure(figsize=(10, 5))
plt.plot(loss_logger.epoch_losses, marker='o')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss per Epoch')
plt.show()
```

Figure 14: Model Training and Evaluation Script

4.11 Confusion Matrix Script

Below code in figure 17 taken from 'pii_detector_using_deberta.ipynb' file which is used to draw confusion metrixand evaluate model performance. Similarily, other models codes

[750/750 32:49, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Precision	Recall	F1
1	0.014100	0.016093	0.996007	0.998001	0.997003
2	0.012100	0.014089	0.997356	0.998079	0.997280
3	0.010000	0.013920	0.997231	0.998070	0.997386
4	0.007200	0.013520	0.997159	0.998055	0.997447
5	0.008600	0.013534	0.997105	0.998001	0.997431

Figure 15: Train- Eval Result

```
Classification Report:
                   precision    recall  f1-score   support

          B-EMAIL       0.00      0.00      0.00         7
    I-URL_PERSONAL       0.00      0.00      0.00         0
  B-STREET_ADDRESS       0.00      0.00      0.00         0
      I-PHONE_NUM        0.00      0.00      0.00         9
      B-USERNAME         0.00      0.00      0.00         0
        B-ID_NUM         0.00      0.00      0.00        34
               O         1.00      1.00      1.00    204733
    I-NAME_STUDENT       0.41      0.14      0.21       151
    B-URL_PERSONAL       0.64      1.00      0.78        16
  I-STREET_ADDRESS       0.00      0.00      0.00         0
    B-NAME_STUDENT       0.42      0.13      0.20       190
      B-PHONE_NUM        0.00      0.00      0.00         3

          accuracy                           1.00    205143
         macro avg       0.21      0.19      0.18    205143
      weighted avg       1.00      1.00      1.00    205143
```

Figure 16: Classification Report

are there in their respective ipynb file which is used for fine tuning models for sensitive data detection and the result of function are in figure 15 and figure 16.

```
In [29]:
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Get predictions from the trainer
predictions, labels, _ = trainer.predict(tokenized_datasets['test'])

# Step 2: Convert predictions to label IDs
predicted_label_ids = np.argmax(predictions, axis=2)

# Step 3: Map IDs to actual label names
true_labels = [
    [id2label[label] for label in doc if label != -100]  # Ignore padding tokens
    for doc in labels
]
pred_labels = [
    [id2label[pred] for pred, label in zip(doc, labels[i]) if label != -100]
    for i, doc in enumerate(predicted_label_ids)
]

# Step 4: Flatten the lists for confusion matrix
true_labels_flat = [label for sublist in true_labels for label in sublist]
pred_labels_flat = [label for sublist in pred_labels for label in sublist]

# Step 5: Generate confusion matrix
unique_labels = list(set(true_labels_flat + pred_labels_flat))  # Get all unique labels
cm = confusion_matrix(true_labels_flat, pred_labels_flat, labels=unique_labels)

# Step 6: Plot the confusion matrix
plt.figure(figsize=(12, 10))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=unique_labels)
disp.plot(cmap="viridis", xticks_rotation=45)
plt.title("Confusion Matrix for NER (DeBERTa)")
plt.show()
```

Figure 17: Confusion Metrics Script

4.12 Model Inference

Below code in figure 18 is taken from 'sensitive_data_inference_using_all_models.ipynb' file which is used to inference the Longformer fine-tuned model and the figure 19 is the output of the same. Similarily, other models' codes are there in that file too for inference and fined-tuned model for DistilBERT, DeBERTa, RoBERTa, and Longformer are available on shared google drive [6]. to get download and upload in ones drive to access the model inference result too.

[6] https://drive.google.com/drive/folders/1OhDmoqP-g6Xn2DqGkGqSx2zTErDQa0Og?usp=sharing

x23194545_Sensitive_data_detection / sensitive_data_inference_using_all_models.ipynb

Preview | Code | Blame | 524 lines (524 loc) · 22.7 KB | Code 55% faster with GitHub Copilot | Raw

Inference with Longformer

In [5]:

```
from transformers import AutoTokenizer, AutoModelForTokenClassification
import torch

# Specify the saved directory
save_directory = "/content/drive/My Drive/PII_models/longformer_sensitive_data_detection_model"

# Load the tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(save_directory)
model = AutoModelForTokenClassification.from_pretrained(save_directory)

# Move model to the appropriate device (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

print("Model and tokenizer loaded successfully!")

# Input text for inference
text = "Sensitive data like credit card numbers, addresses, or personal emails such as john.doe@gmail.com should be protect

# Tokenize the input
tokens = tokenizer(
    text,
    return_tensors="pt",
    padding=True,
    truncation=True,
    max_length=512,  # Adjust based on your model's maximum length
    is_split_into_words=False
```

Figure 18: Longformer Model Inference Code Sample

```
Model and tokenizer loaded successfully!

Cleaned Predictions:
Sensitive: O
data: O
like: O
credit: O
card: O
numbers,: O
addresses,: O
or: O
personal: O
emails: O
such: O
as: O
john.doe@gmail.com: B-EMAIL
should: O
be: O
protected.: O
```

Figure 19: Inference Result