In [ ]:

```
!pip install transformers -U
```

In [ ]:

```
import pandas as pd
```

In [ ]:

```
# https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge
data = pd.read_csv("/content/train.csv", engine="python")
data.head()
```

In [ ]:

```
data['toxic'].value_counts()
```

In [ ]:

```
data = data[['comment_text','toxic']]
data = data[0:1000]
data.head()
```

In [ ]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
import torch
from transformers import TrainingArguments, Trainer
from transformers import BertTokenizer, BertForSequenceClassification
```

In [ ]:

```
from transformers import BertTokenizer, BertForSequenceClassification
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',num_label
```

In [ ]:

```
model
```

In [ ]:

```
# model = model.to('cuda')
```

In [ ]:

```
sample_data = ["I am eating","I am playing "]
tokenizer(sample_data, padding=True, truncation=True, max_length=512)
```

In [ ]:

```
X = list(data["comment_text"])
y = list(data["toxic"])
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,stratify=y)
X_train_tokenized = tokenizer(X_train, padding=True, truncation=True, max_length=51
X_val_tokenized = tokenizer(X_val, padding=True, truncation=True, max_length=512)
```

In [ ]:

```
X_train_tokenized.keys()
```

In [ ]:

```
print(X_train_tokenized['attention_mask'][0])
```

In [ ]:

```
len(X_train),len(X_val)
```

In [ ]:

```
# Create dataset
class Dataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels
    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        if self.labels:
            item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.encodings["input_ids"])
```

In [ ]:

```
train_dataset = Dataset(X_train_tokenized, y_train)
val_dataset = Dataset(X_val_tokenized, y_val)
```

In [ ]:

```
train_dataset[5]
```

In [ ]:

```
def compute_metrics(p):
    print(type(p))
    pred, labels = p
    pred = np.argmax(pred, axis=1)

    accuracy = accuracy_score(y_true=labels, y_pred=pred)
    recall = recall_score(y_true=labels, y_pred=pred)
    precision = precision_score(y_true=labels, y_pred=pred)
    f1 = f1_score(y_true=labels, y_pred=pred)

    return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f
```

In [ ]:

```
# Define Trainer
args = TrainingArguments(
    output_dir="output",
    num_train_epochs=1,
    per_device_train_batch_size=8

)
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics
)
```

In [ ]:

```
trainer.train()
```

In [ ]:

```
trainer.evaluate()
```

In [ ]:

```
np.set_printoptions(suppress=True)
```

In [ ]:

```
text = "That was good point"
# text = "go to hell"
inputs = tokenizer(text,padding = True, truncation = True, return_tensors='pt').to(
outputs = model(**inputs)
print(outputs)
predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
print(predictions)
```

```
predictions = predictions.cpu().detach().numpy()
if predictions[0][0] > 0.5:
  print("Not Toxic")
else:
  print("Toxic")
```

In [ ]:

```
trainer.save_model('CustomModel')
```

In [ ]:

```
model_2 = BertForSequenceClassification.from_pretrained("CustomModel")
model_2.to('cuda')
```

In [ ]:

```
# text = "That was good point"
inputs = tokenizer(text,padding = True, truncation = True, return_tensors='pt').to(
outputs = model_2(**inputs)
predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)
predictions = predictions.cpu().detach().numpy()
if predictions[0][0] > 0.5:
  print("Not Toxic")
else:
  print("Toxic")
```

In [ ]: