PII Anonymization Tool

This notebook demonstrates a named entity recognition (NER) system for detecting and anonymizing personally identifiable information (PII) in text.

Till the end:

1. Train a model to recognize named entities (if needed)
2. Use the model to anonymize text by replacing entities
3. Try different anonymization styles
4. Interactively use the system through a web interface

1. Setup and Installation

```
def train_and_test_model(output_dir="./model", force_train=False):
    #model existance check, will not train if already exists.
    exists, is_valid = check_model_exists(output_dir)

    if exists and is_valid and not force_train:
        print(f"✓ Valid trained model already exists at {output_dir}")
        print("  Skipping training to save time.")
    else:
        #data loading and preparing
        print("\n=== Data Loading and Preparation ===")
        processor = PIIDataProcessor()
        datasets = processor.load_conll2003_dataset()

        # Debugging:
        print("Available columns in the dataset:")
        print(datasets["train"].column_names)

        datasets = processor.convert_to_ner_format(datasets)
```

```
from datasets import load_dataset, disable_caching
# Disable caching for this run or clear existing cache
disable_caching()
```

```
# Install required packages
!pip install transformers datasets seqeval flask gradio
!pip install --upgrade transformers
```

```
    Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packag
```

```
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from s
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from pytho
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from mar
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.53.0
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from trans
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from tr
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from tr
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from trans
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from tra
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from r
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (
```

2. Data Processor

Next, let's create the data processor class that will handle loading and processing the CoNLL-2003 dataset.

```
!wget https://github.com/panchalaman/PII-Anonymization-System/raw/refs/heads/main/AI%20Applications
```

```
--2025-07-03 10:32:02--  https://github.com/panchalaman/PII-Anonymization-System/raw/refs/heads/
Resolving github.com (github.com)... 140.82.116.3
Connecting to github.com (github.com)|140.82.116.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/panchalaman/PII-Anonymization-System/refs/heads/mai
--2025-07-03 10:32:02--  https://raw.githubusercontent.com/panchalaman/PII-Anonymization-System/
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... conn
HTTP request sent, awaiting response... 200 OK
Length: 982975 (960K) [application/zip]
Saving to: ‘conll2003.zip.1’

conll2003.zip.1     100%[===================>] 959.94K  --.-KB/s    in 0.04s

2025-07-03 10:32:02 (24.6 MB/s) - ‘conll2003.zip.1’ saved [982975/982975]
```

```
from typing import Dict, List
from datasets import load_dataset

class PIIDataProcessor:
    def __init__(self):
        self.label_list = [
            "O",  # Not an entity
            "B-PER", "I-PER",  # Person entities
            "B-ORG", "I-ORG",  # Organization entities
            "B-LOC", "I-LOC",  # Location entities
            "B-MISC", "I-MISC"  # Miscellaneous entities
        ]

        # Create mappings between labels and IDs
        self.label2id = {label: i for i, label in enumerate(self.label_list)}
        self.id2label = {i: label for i, label in enumerate(self.label_list)}

    def load_conll2003_dataset(self):
        print("Loading CoNLL-2003 dataset...")
        dataset = load_dataset("conll2003")
        print(f"Dataset loaded with {len(dataset['train'])} training, "
              f"{len(dataset['validation'])} validation, and "
              f"{len(dataset['test'])} test examples")
        return dataset

    def convert_to_ner_format(self, dataset):
        return dataset
```

3. Tokenizer

Now, let's create the tokenizer class that handles subword tokenization and aligns entity labels.

```
class PIITokenizer:


    def __init__(self, tokenizer, label2id: Dict[str, int]):

        self.tokenizer = tokenizer
        self.label2id = label2id
    def tokenize_and_align_labels(self, examples):
        # Tokenize the input words and get word IDs to align labels
        tokenized_inputs = self.tokenizer(
            examples["tokens"],  # List of tokens for each example
            truncation=True,  # Truncate to max length if needed
            is_split_into_words=True,  # Input is already split into words
            padding='max_length',  # Pad to max length
            max_length=128,  # Maximum sequence length
            return_tensors="pt"  # Return PyTorch tensors
        )

        # Align labels with subword tokens
        labels = []
        for i, label in enumerate(examples["ner_tags"]):
            # Get word IDs for current example
            word_ids = tokenized_inputs.word_ids(batch_index=i)

            # Track previous word to handle subword tokens
            previous_word_idx = None
            label_ids = []

            # For each token in the sequence:
            for word_idx in word_ids:
                # Special tokens have word_idx = None
                if word_idx is None:
                    label_ids.append(-100)  # Ignore special tokens for loss

                # If this is a new word (not a subword continuation)
                elif word_idx != previous_word_idx:
                    label_ids.append(label[word_idx])  # Use actual label

                # If this is a subword continuation
                else:
                    label_ids.append(-100)  # Ignore subword continuations

                # Update the previous word index
                previous_word_idx = word_idx

            labels.append(label_ids)

        # Add aligned labels to tokenized inputs
        tokenized_inputs["labels"] = labels
        return tokenized_inputs
```

4. Metrics

Now, let's create the metrics class that will evaluate the model performance.

```
import numpy as np
from seqeval.metrics import f1_score, precision_score, recall_score, accuracy_score

class PIIMetrics:
    """
    Computes metrics for evaluating NER model performance.
    """

    def __init__(self, label_list: List[str]):
        """
        Initialize the metrics calculator with the list of possible labels.
        """
        self.label_list = label_list

    def compute_metrics(self, eval_prediction):
        """
        Compute evaluation metrics from model predictions.
        """
        # Unpack predictions and labels
        predictions, labels = eval_prediction

        # Get predicted label IDs (argmax along the class dimension)
        predictions = np.argmax(predictions, axis=2)

        # Convert predictions to label names, ignoring special tokens (-100)
        true_predictions = [
            [self.label_list[p] for (p, l) in zip(prediction, label) if l != -100]
            for prediction, label in zip(predictions, labels)
        ]

        # Convert reference labels to label names, ignoring special tokens (-100)
        true_labels = [
            [self.label_list[l] for (p, l) in zip(prediction, label) if l != -100]
            for prediction, label in zip(predictions, labels)
        ]

        # Compute seqeval metrics
        return {
            "accuracy": accuracy_score(true_labels, true_predictions),
            "f1": f1_score(true_labels, true_predictions),
            "precision": precision_score(true_labels, true_predictions),
            "recall": recall_score(true_labels, true_predictions),
        }
```

5. Trainer

Now, let's create the trainer class that will train and evaluate the model.

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

class PIITrainer:
    """
    Trainer for the NER model used for PII detection.
    """

    def __init__(self, model_name: str, datasets, label_list: List[str],
                 id2label: Dict, label2id: Dict, output_dir: str = "./pii-model"):
        """
        Initialize the trainer with model and data.
        """
        self.model_name = model_name
        self.output_dir = output_dir

        # Initialize tokenizer and model
        print(f"Initializing tokenizer and model from {model_name}...")
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.pii_tokenizer = PIITokenizer(self.tokenizer, label2id)

        # Tokenize the datasets
        print("Tokenizing datasets...")
        self.tokenized_datasets = datasets.map(
            self.pii_tokenizer.tokenize_and_align_labels,
            batched=True,
            remove_columns=datasets["train"].column_names
        )

        # Initialize model with the correct number of labels
        print(f"Initializing model with {len(label_list)} labels...")
        self.model = AutoModelForTokenClassification.from_pretrained(
            model_name,
            num_labels=len(label_list),
            id2label=id2label,
            label2id=label2id
        )

        # Set up metrics calculator
        self.metrics = PIIMetrics(label_list)

        # Configure training arguments
        use_cuda = torch.cuda.is_available()
        device_str = "CUDA" if use_cuda else "CPU"
        print(f"Using {device_str} for training")



        self.training_args = TrainingArguments(
            output_dir=output_dir,
            eval_strategy="epoch",
            save_strategy="epoch",
            learning_rate=2e-5,
            per_device_train_batch_size=16,
            per_device_eval_batch_size=16,
            num_train_epochs=3,
            weight_decay=0.01,
            fp16=use_cuda,  # Use FP16 if CUDA is available
            load_best_model_at_end=True,
            metric_for_best_model="f1",
            report_to="none",
            logging_dir='./logs',
            logging_steps=100,
        )

        # Initialize the trainer
        self.trainer = Trainer(
            model=self.model,
            args=self.training_args,
            train_dataset=self.tokenized_datasets["train"],
            eval_dataset=self.tokenized_datasets["validation"],
            data_collator=DataCollatorForTokenClassification(self.tokenizer),
            compute_metrics=self.metrics.compute_metrics,
        )

    def train(self):
        """Train the model."""
        print("Starting model training...")
        return self.trainer.train()

    def evaluate(self):
        """Evaluate the model on the validation set."""
        print("Evaluating model...")
        return self.trainer.evaluate()

    def save_model(self):
        """Save the model, tokenizer, and configuration files."""
        print(f"Saving model to {self.output_dir}...")
        self.trainer.save_model(self.output_dir)
        self.tokenizer.save_pretrained(self.output_dir)
        self.model.config.save_pretrained(self.output_dir)
        print("Model saved successfully!")
```

6. Anonymizer

Next, let's create the anonymizer class that will use the trained model to detect and anonymize entities.

```
import os
import json
from transformers import pipeline

class PIIAnonymizer:
    """
    Anonymizes personally identifiable information in text.
    """

    def __init__(self, model_path: str, device: str = None):
        """
        Initialize the anonymizer with a trained model.
        """
        # Set device (use CUDA if available and not explicitly set to CPU)
        if device is None:
            self.device = "cuda" if torch.cuda.is_available() else "cpu"
        else:
            self.device = device

        print(f"Loading model from: {model_path}")
        print(f"Device set to use {self.device}")
        try:
            # Try to load from local directory
            if os.path.exists(model_path) and os.path.isdir(model_path):
                # Check for required files
                if not os.path.exists(os.path.join(model_path, "config.json")):
                    print(f"Warning: config.json not found in {model_path}")
                    print("Directory contents:")
                    print(os.listdir(model_path))

                # Load model and tokenizer
                self.tokenizer = AutoTokenizer.from_pretrained(model_path)
                self.model = AutoModelForTokenClassification.from_pretrained(model_path).to(self.de

                # Load label mapping (if available)
                if os.path.exists(os.path.join(model_path, "id2label.json")):
                    with open(os.path.join(model_path, "id2label.json"), "r") as f:
                        self.id2label = json.load(f)
                else:
                    # Fall back to model's config
                    self.id2label = self.model.config.id2label
            else:
                # If local path doesn't exist, try using it as a model ID from HuggingFace Hub
                print(f"Model directory {model_path} not found, attempting to load from HuggingFace
                self.tokenizer = AutoTokenizer.from_pretrained(model_path)
                self.model = AutoModelForTokenClassification.from_pretrained(model_path).to(self.de
                self.id2label = self.model.config.id2label

            # Create NER pipeline for easy inference
            self.nlp = pipeline(
                "ner",
                model=self.model,
                tokenizer=self.tokenizer,
                device=0 if self.device == "cuda" else -1,
                aggregation_strategy="simple"  # Merge subword tokens
            )
            print(f"Successfully loaded model and created pipeline")

        except Exception as e:
            print(f"Error loading model: {str(e)}")
            raise

    def anonymize_text(self, text: str, style: str = "tag") -> str:
        """
        Anonymize PII in text by replacing entities with placeholders.
        """
        if not text:
            return ""

        # Detect entities
        entities = self.nlp(text)

        # Process in reverse order to avoid index shifting
        anonymized_text = text
        for entity in reversed(entities):
            if entity['entity_group'] != 'O':
                start, end = entity["start"], entity["end"]

                # Extract the entity type (remove B-, I- prefixes)
                tag = entity['entity_group'].split('-')[-1] if '-' in entity['entity_group'] else e

                # Apply anonymization based on style
                if style == "tag":
                    replacement = f"[{tag}]"
                elif style == "mask":
                    replacement = "X" * (end - start)
                else:  # redact
```

```
                    replacement = "[REDACTED]"

                # Replace entity in text
                anonymized_text = anonymized_text[:start] + replacement + anonymized_text[end:]

        return anonymized_text

    def detect_entities(self, text: str, threshold: float = 0.7) -> list:
        """
        Detect entities in text with improved boundary detection.
        """
        # Simple implementation that just uses the pipeline
        if not text:
            return []

        # Get entities from pipeline
        entities = self.nlp(text)

        # Convert to more user-friendly format
        result = []
        for entity in entities:
            if entity['score'] >= threshold:
                # Extract entity type without B-/I- prefix
                entity_type = entity['entity_group'].split('-')[-1] if '-' in entity['entity_group'

                result.append({
                    "type": entity_type,
                    "text": entity['word'],
                    "confidence": entity['score']
                })

        return result
```

7. Utility Functions

Let's create utility functions for model management and testing.

```
import shutil

def save_model_metadata(output_dir, processor):
    """Save additional metadata files needed for the model."""
    print(f"Saving extra model files to {output_dir}")

    # Save label list
    with open(os.path.join(output_dir, "label_list.txt"), "w") as f:
        f.write("\n".join(processor.label_list))

    # Save id2label mapping
    with open(os.path.join(output_dir, "id2label.json"), "w") as f:
        json.dump(processor.id2label, f)

    # Save label2id mapping
    with open(os.path.join(output_dir, "label2id.json"), "w") as f:
        json.dump(processor.label2id, f)


def check_model_exists(model_dir):
    """Check if a trained model exists and is valid."""
    # Check if directory exists
    if not os.path.exists(model_dir):
        return False, False

    # Check if directory is not empty
    if not os.listdir(model_dir):
        return True, False

    # Check for essential files
    required_files = ["config.json", "pytorch_model.bin"]
    for file in required_files:
        if not os.path.exists(os.path.join(model_dir, file)):
            return True, False

    return True, True


def try_fallback_model(test_sentences):
    """Try using a fallback model if the custom model fails."""
    print("\nFallback: Loading a pre-trained NER model from HuggingFace instead")
    try:
        fallback_anonymizer = PIIAnonymizer(
            model_path="dslim/bert-base-NER",
            device="cuda" if torch.cuda.is_available() else "cpu"
        )
        print("\n===== FALLBACK ANONYMIZATION RESULTS =====\n")
        for sentence in test_sentences:
            anonymized_sentence = fallback_anonymizer.anonymize_text(sentence)
            print(f"Original:   {sentence}")
            print(f"Anonymized: {anonymized_sentence}")
            print("-" * 50)
    except Exception as fallback_error:
        print(f"\nFallback also failed: {str(fallback_error)}")
        print("Please check your installation of transformers and ensure you have internet connecti


def get_test_sentences():
    """Get a list of test sentences for demonstration."""
    return [
        "EU rejects German call to boycott British lamb.",
        "Peter Blackburn works at Microsoft in Seattle.",
        "BRUSSELS 1996-08-22 - The meeting took place at the Grand Hotel.",
        "John Smith and Sarah Johnson attended the conference in New York City last week.",
        "The European Commission said on Thursday it disagreed with German advice to consumers to sl
        "Apple Inc. announced its new headquarters in Cupertino, California."
    ]
```

8. Main Training and Testing Function

Now, let's create a function that trains the model and tests it.

```
def train_and_test_model(output_dir="./model", force_train=False):
    """Train (if needed) and test a PII anonymization model."""
    # --- 1. Check if model already exists ---
    exists, is_valid = check_model_exists(output_dir)

    if exists and is_valid and not force_train:
        print(f"✓ Valid trained model already exists at {output_dir}")
        print("  Skipping training to save time.")
        print("  (Use force_train=True to retrain anyway)")
    else:
        if exists and not is_valid:
            print(f"! Found model directory at {output_dir} but it's incomplete or invalid")
            print("  Removing directory and retraining...")
            shutil.rmtree(output_dir)
            os.makedirs(output_dir, exist_ok=True)
        elif force_train:
            print(f"! Forcing retraining as requested")
            shutil.rmtree(output_dir, ignore_errors=True)
            os.makedirs(output_dir, exist_ok=True)
        else:
            print(f"! No existing model found at {output_dir}")
            os.makedirs(output_dir, exist_ok=True)

        # --- 2. Data Loading and Preparation ---
        print("\n=== Data Loading and Preparation ===")
        processor = PIIDataProcessor()
        datasets = processor.load_conll2003_dataset()
        datasets = processor.convert_to_ner_format(datasets)

        # --- 3. Training ---
        print("\n=== Model Training ===")
        trainer = PIITrainer(
            model_name="distilbert-base-uncased",
            datasets=datasets,
            label_list=processor.label_list,
            id2label=processor.id2label,
            label2id=processor.label2id,
            output_dir=output_dir
        )

        trainer.train()

        # --- 4. Evaluation ---
        print("\n=== Model Evaluation ===")
        results = trainer.evaluate()
        print(f"Evaluation results: {results}")

        # --- 5. Save Model ---
        print("\n=== Saving Model ===")
        trainer.save_model()
        save_model_metadata(output_dir, processor)
        print(f"Model saved to {output_dir}")

    # --- 6. Test Anonymization ---
    print("\n=== Testing Anonymization ===")
    test_sentences = get_test_sentences()

    try:
        print("Initializing anonymizer...")
        anonymizer = PIIAnonymizer(
            model_path=output_dir,
            device="cuda" if torch.cuda.is_available() else "cpu"
        )

        print("\n===== ANONYMIZATION RESULTS =====\n")
        for sentence in test_sentences:
            anonymized_sentence = anonymizer.anonymize_text(sentence)
            print(f"Original:   {sentence}")
            print(f"Anonymized: {anonymized_sentence}")
            print("-" * 50)

        return anonymizer

    except Exception as e:
        print(f"\nError in anonymization: {str(e)}")
        try_fallback_model(test_sentences)
        return None
```

9. Run Training and Testing

Now, let's train the model (if needed) and test the anonymization functionality.

```
import shutil

# Clear the Hugging Face dataset cache manually
shutil.rmtree("./hf_cache", ignore_errors=True)
```

```
pip install --upgrade datasets
```

```
Requirement already satisfied: datasets in /usr/local/lib/python3.11/dist-packages (3.6.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from dataset
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from data
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from dat
Requirement already satisfied: xxhash in /usr/local/lib/python3.11/dist-packages (from datasets)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from datase
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from data
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from req
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from par
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from p
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from ai
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-
```

```
# Train and test the model (will skip training if valid model exists)
MODEL_DIR = "/content/pii-model"
anonymizer = train_and_test_model(MODEL_DIR, force_train=False)
```

```
! No existing model found at /content/pii-model

=== Data Loading and Preparation ===
Loading CoNLL-2003 dataset...
/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://hugging
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 dat
  warnings.warn(
README.md:      12.3k/? [00:00<00:00, 1.16MB/s]

conll2003.py:      9.57k/? [00:00<00:00, 798kB/s]
Dataset loaded with 14041 training, 3250 validation, and 3453 test examples

=== Model Training ===
Initializing tokenizer and model from distilbert-base-uncased...
tokenizer_config.json: 100%                    48.0/48.0 [00:00<00:00, 3.99kB/s]

config.json: 100%                    483/483 [00:00<00:00, 32.7kB/s]

vocab.txt: 100%                    232k/232k [00:00<00:00, 15.2MB/s]

tokenizer.json: 100%                    466k/466k [00:00<00:00, 6.92MB/s]
Tokenizing datasets...
Map: 100%                    14041/14041 [00:03<00:00, 2765.94 examples/s]

Map: 100%                    3250/3250 [00:00<00:00, 4607.19 examples/s]

Map: 100%                    3453/3453 [00:00<00:00, 4911.11 examples/s]
Initializing model with 9 labels...
model.safetensors: 100%                    268M/268M [00:02<00:00, 139MB/s]
Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint
You should probably TRAIN this model on a down-stream task to be able to use it for predictions
Using CUDA for training
Starting model training...
[2634/2634 03:42, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.066700	0.056248	0.983211	0.911494	0.905736	0.917326
2	0.041000	0.048500	0.986775	0.931894	0.926160	0.937700
3	0.022400	0.049768	0.987165	0.934304	0.928785	0.939889

```
=== Model Evaluation ===
Evaluating model...
[204/204 00:03]
Evaluation results: {'eval_loss': 0.04976791515946388, 'eval_accuracy': 0.9871647546890521, 'eva

=== Saving Model ===
Saving model to /content/pii-model...
Device set to use cuda:0
Model saved successfully!
Saving extra model files to /content/pii-model
Model saved to /content/pii-model

=== Testing Anonymization ===
Initializing anonymizer...
Loading model from: /content/pii-model
Device set to use cuda
Successfully loaded model and created pipeline

===== ANONYMIZATION RESULTS =====

Original:   EU rejects German call to boycott British lamb.
Anonymized: [ORG] rejects [MISC] call to boycott [MISC] lamb.
--------------------------------------------------
Original:   Peter Blackburn works at Microsoft in Seattle.
Anonymized: [PER] works at [ORG] in [LOC]
```

```
Anonymized: [PER] works at [ORG] in [LOC].
--------------------------------------------------
Original:   BRUSSELS 1996-08-22 - The meeting took place at the Grand Hotel.
Anonymized: [LOC] 1996-08-22 - The meeting took place at the [LOC].
--------------------------------------------------
Original:   John Smith and Sarah Johnson attended the conference in New York City last week.
Anonymized: [PER] and [PER] attended the conference in [LOC] last week.
--------------------------------------------------
Original:   The European Commission said on Thursday it disagreed with German advice to consume
Anonymized: The [ORG] said on Thursday it disagreed with [MISC] advice to consumers to shun [MIS
--------------------------------------------------
Original:   Apple Inc. announced its new headquarters in Cupertino, California.
Anonymized: [ORG]. announced its new headquarters in [LOC][LOC], [LOC].
--------------------------------------------------
```

10. Try Different Anonymization Styles

Let's try the different anonymization styles available.

Double-click (or enter) to edit

```python
# Function to demonstrate different anonymization
def demo_anonymization_styles(text):
    if anonymizer is None:
        print("Anonymizer not available. Please check previous steps.")
        return

    print(f"Original text: {text}\n")

    print("Tag style (default):")
    print(anonymizer.anonymize_text(text, style="tag"))
    print("\nMask style:")
    print(anonymizer.anonymize_text(text, style="mask"))
    print("\nRedact style:")
    print(anonymizer.anonymize_text(text, style="redact"))

    print("\nDetected entities:")
    entities = anonymizer.detect_entities(text, threshold=0.6)
    for entity in entities:
        print(f"• {entity['text']} - {entity['type']} ({entity['confidence']:.2%})")

# Try with a custom example
demo_text = "John Smith is the CEO of Acme Corporation based in New York City."
demo_anonymization_styles(demo_text)
```

```
Original text: John Smith is the CEO of Acme Corporation based in New York City.

Tag style (default):
[PER] is the CEO of [ORG] based in [LOC].

Mask style:
XXXXXXXXXX is the CEO of XXXXXXXXXXXXXXXX based in XXXXXXXXXXXXX.

Redact style:
[REDACTED] is the CEO of [REDACTED] based in [REDACTED].

Detected entities:
• john smith - PER (99.79%)
• acme corporation - ORG (98.61%)
• new york city - LOC (99.19%)
```

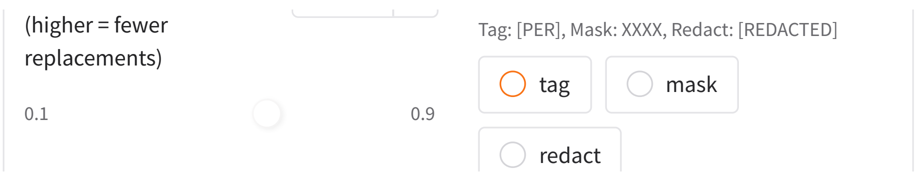
11. Interactive Demo with Gradio

Now, let's create an interactive web interface using Gradio.

```
# Install gradio if not already installed
!pip install -q gradio
```

```
import gradio as gr

def create_gradio_interface(anon):
    """
    Create a Gradio interface for PII anonymization.
    """
    if anon is None:
        print("Anonymizer not available. Can't create interface.")
        return None

    def process_text(text, threshold, style, show_entities):
        """Process text for the Gradio interface."""
        if not text:
            return "Please enter some text to anonymize."

        try:
            # Detect entities
            entities = anon.detect_entities(text, threshold)

            # Anonymize text
            anonymized = anon.anonymize_text(text, style)

            # Return result with or without entity details
            if show_entities:
                entity_list = "\n".join([
                    f"• {e['text']} - {e['type']} ({e['confidence']:.2%})"
                    for e in entities
                ])
                return f"Anonymized text:\n{anonymized}\n\nDetected entities:\n{entity_list}"
            else:
                return anonymized

        except Exception as e:
            import traceback
            return f"Error processing text: {str(e)}\n\n{traceback.format_exc()}"

    def highlight_entities(text, threshold=0.7):
        """Generate HTML with highlighted entities for visualization."""
        if not text:
            return ""

        # Detect entities
        entities = anon.detect_entities(text, threshold)

        # Define entity type colors
        colors = {
            "PER": "#ffcccc",  # Light red for persons
            "ORG": "#ccffcc",  # Light green for organizations
            "LOC": "#ccccff",  # Light blue for locations
            "MISC": "#ffffcc"  # Light yellow for miscellaneous
        }

        # Find entity positions in text
        entities_with_pos = []
        for entity in entities:
            start = text.find(entity["text"])
            if start >= 0:
                entities_with_pos.append({
                    "start": start,
                    "end": start + len(entity["text"]),
                    "type": entity["type"],
                    "confidence": entity["confidence"]
                })

        # Sort by position reversed (to avoid index shifting)
        entities_with_pos.sort(key=lambda x: x["start"], reverse=True)

        # Insert HTML tags for highlighting
        html_text = text
        for entity in entities_with_pos:
            entity_text = text[entity["start"]:entity["end"]]
            entity_type = entity["type"]
            confidence = entity["confidence"]
            color = colors.get(entity_type, "#eeeeee")

            # Create highlighted span
            html_entity = (
                f'<span style="background-color: {color};" '
                f'title="{entity_type} ({confidence:.1%})">{entity_text}</span>'
            )

            # Replace text with highlighted version
            html_text = html_text[:entity["start"]] + html_entity + html_text[entity["end"]:]

        return html_text

    # Create interface with tabs for different functionalities
    with gr.Blocks(title="PII Anonymization Tool") as demo:
        gr.Markdown("# PII Anonymization Tool")
        gr.Markdown("""
        This tool automatically detects and anonymizes personally identifiable information in text
        using a fine-tuned Named Entity Recognition model.

        It can detect names, organizations, locations, and other entities in your text.
        """)

        with gr.Tab("Text Anonymization"):
            with gr.Row():
                with gr.Column():
                    # Input area
                    input_text = gr.Textbox(
                        lines=5,
                        placeholder="Enter text to anonymize (e.g., 'John Smith works at Microsoft ir
                        label="Input Text"
                    )

                    with gr.Row():
                        # Configuration options
                        threshold = gr.Slider(
                            minimum=0.1,
                            maximum=0.9,
                            value=0.6,
                            step=0.05,
                            label="Confidence Threshold (higher = fewer replacements)"
                        )
                        style = gr.Radio(
                            ["tag", "mask", "redact"],
                            label="Anonymization Style",
                            value="tag",
                            info="Tag: [PER], Mask: XXXX, Redact: [REDACTED]"
                        )

                    show_entities = gr.Checkbox(
                        label="Show detected entities",
                        value=True
                    )

                    anonymize_btn = gr.Button("Anonymize Text")

                with gr.Column():
                    # Output area
                    output_text = gr.Textbox(label="Result", lines=10)

            # Example inputs
            examples = gr.Examples(
                examples=[
                    ["John Smith works at Microsoft in New York.", 0.6, "tag", True],
                    ["Please contact Sarah Johnson at sarah.j@example.com or call 555-123-4567.", 0.5
                    ["Patient #12345 was admitted on January 15th with Dr. Williams supervising.", 0.
                    ["EU rejects German call to boycott British lamb.", 0.6, "tag", True],
                    ["Apple Inc. announced its new headquarters in Cupertino, California.", 0.7, "tag
                ],
                inputs=[input_text, threshold, style, show_entities]
            )

            # Connect button to processing function
            anonymize_btn.click(
                fn=process_text,
                inputs=[input_text, threshold, style, show_entities],
                outputs=output_text
            )

        with gr.Tab("Entity Highlighting"):
            with gr.Row():
                with gr.Column():
                    # Input area
                    highlight_input = gr.Textbox(
                        lines=5,
                        placeholder="Enter text to highlight entities",
                        label="Input Text"
                    )
                    highlight_threshold = gr.Slider(
                        minimum=0.1,
                        maximum=0.9,
                        value=0.6,
                        step=0.05,
                        label="Confidence Threshold"
                    )
                    highlight_btn = gr.Button("Highlight Entities")
                with gr.Column():
                    # Output HTML with highlighting
                    highlighted_output = gr.HTML(label="Highlighted Text")

            # Connect button to highlighting function
            highlight_btn.click(
                fn=highlight_entities,
                inputs=[highlight_input, highlight_threshold],
                outputs=highlighted_output
            )

            # Example inputs for highlighting
            highlight_examples = gr.Examples(
                examples=[
                    ["John Smith works at Microsoft in New York.", 0.6],
                    ["EU rejects German call to boycott British lamb.", 0.6],
                    ["The European Commission said on Thursday it disagreed with German advice to con
                ],
                inputs=[highlight_input, highlight_threshold]
            )

        with gr.Tab("About"):
        with gr.Tab("About"):
            # About tab with information about the project
            gr.Markdown("""
            ## About This Tool

            This PII anonymization tool uses a fine-tuned Named Entity Recognition (NER) model
            based on DistilBERT to identify and anonymize personally identifiable information in text

            ### Entity Types
            - **PER**: Person names (e.g., "John Smith")
            - **ORG**: Organizations (e.g., "Microsoft", "European Commission")
            - **LOC**: Locations (e.g., "New York", "Brussels")
            - **MISC**: Miscellaneous entities (e.g., "German", "British")

            ### Anonymization Styles
            - **Tag**: Replaces entities with their type in brackets, e.g., [PER]
            - **Mask**: Replaces characters with 'X', maintaining the entity length
            - **Redact**: Replaces entities with [REDACTED], regardless of entity type or length

            ### Model Information
            This tool uses a model fine-tuned on CoNLL-2003 dataset for named entity recognition.
            """)

    return demo

# Create and launch the interface
if anonymizer is not None:
    demo = create_gradio_interface(anonymizer)
    if demo is not None:
        demo.launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://464aab670116a7f422.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio dep
```


(higher = fewer replacements)

0.1 0.9

Tag: [PER], Mask: XXXX, Redact: [REDACTED]

Reference:

@inproceedings{tjong-kim-sang-de-meulder-2003-introduction, title = "Introduction to the {C}o{NLL}-2003 Shared Task: Language-Independent Named Entity Recognition", author = "Tjong Kim Sang, Erik F. and De Meulder, Fien", booktitle = "Proceedings of the Seventh Conference on Natural Language Learning at {HLT}-{NAACL} 2003", year = "2003", url = "https://www.aclweb.org/anthology/W03-0419", pages = "142--147", }

```
!jupyter nbconvert --to html /content/PII-Anonymization-Notebook.ipynb
```

```
                copy of reveal.js. e.g., "reveal.js".
                If a relative path is given, it must be a subdirectory of the
                current directory (from which the server is run).
                See the usage documentation
                (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-slideshow)
                for more details.
        Default: ''
        Equivalent to: [--SlidesExporter.reveal_url_prefix]
    --nbformat=<Enum>
        The nbformat version to write.
                Use this to downgrade notebooks.
        Choices: any of [1, 2, 3, 4]
        Default: 4
        Equivalent to: [--NotebookExporter.nbformat_version]

    Examples
    --------

        The simplest way to use nbconvert is

                > jupyter nbconvert mynotebook.ipynb --to html

                Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'p

                > jupyter nbconvert --to latex mynotebook.ipynb

                Both HTML and LaTeX support multiple output templates. LaTeX includes
                'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
                'classic'. You can specify the flavor of the format used.

                > jupyter nbconvert --to html --template lab mynotebook.ipynb

                You can also pipe the output to stdout, rather than a file

                > jupyter nbconvert mynotebook.ipynb --stdout

                PDF is generated via latex

                > jupyter nbconvert mynotebook.ipynb --to pdf

                You can get (and serve) a Reveal.js-powered slideshow

                > jupyter nbconvert myslides.ipynb --to slides --post serve

                Multiple notebooks can be given at the command line in a couple of
                different ways:

                > jupyter nbconvert notebook*.ipynb
                > jupyter nbconvert notebook1.ipynb notebook2.ipynb
```