## Cybersecurity M

Academic Year 2024-2025

# Dataset Anonymizer

*AI-based system for anonymizing data, ensuring privacy while maintaining reliability for statistical analysis*

Giacomo Vigarelli
Francesco Simoni

# Table of Contents

# 1 Introduction

## 1.1 Overview of Data Anonymization

**Data anonymization** masks personally identifiable information to ensure individuals' privacy. It is crucial especially when data is shared, analysed or used for research purposes. Data protection requires the implementation of security systems to control access to resources (such as databases). Additionally it is essential to ensure that information remains anonymous if a data breach occurs, preventing malicious actors from identifying the individuals' data.
Reliability and integrity of the collected data are preserved with this measures, maintaining the accuracy of the stored information. Moreover, data anonymization facilitates conformity with current european regulations (General Data Protection Regulation, GDPR) and others. They impose strict provisions on the collection, processing, and storage of personal information.
However, only data anonymization alone is not sufficient to eliminate all types of risk. Re-identification attacks and inference attacks can undermine these measures in the absence of adequate countermeasures. It is essential to implement additional safeguards to mitigate these potential threats and ensure comprehensive data protection.

## 1.2 Project Goals

The present project aims to implement a Machine Learning prototype for anonymizing a dataset while ensuring privacy.
The dataset chosen for training the model is MNIST: a large database of handwritten digits. This dataset was selected because it is well-structured, widely used and compatible with our dependencies.
We use PyTorch as the library for training machine learning models, and Opacus to incorporate differential privacy mechanisms by adding noise during training. The main goal is to train multiple ML models with varying levels of differential privacy, defined by the epsilon parameter, as well as a model without differential privacy for comparison.
Accuracy and loss metrics will be analyzed for each epsilon value to highlight how increased privacy protection (lower epsilon) impacts model performance.

# 2 Data Anonymization

## 2.1 Differential Privacy

Differential Privacy is designed to safeguard the privacy of individuals within a dataset. It ensures that the inclusion or exclusion of a single person does not significantly affect the outcomes of analyses, preventing anyone from inferring information about that individual. This is achieved by introducing a controlled level of random "**noise**" into the results of data queries. The added noise is calibrated such that it masks individual contributions without significantly compromising the overall accuracy of the analyses. Thanks to this approach, it is possible to share and analyze sensitive data securely, ensuring that personal information remains protected.

## 2.2 Formal Definition

An algorithm $A$ is *differentially private* if, for every pair of datasets $D$ and $D'$ that differ by a single record, and for every set of possible outputs $S$ of the algorithm, the following holds:

$$\Pr[A(D) \in S] \leq e^{\epsilon} \cdot \Pr[A(D') \in S]$$

where:

- $A$ is the algorithm.
- $\epsilon$ is the **privacy parameter** (epsilon).

Which means that the inclusion or exclusion of a single individual in the dataset does not significantly alter the probability of the algorithm's outputs, ensuring that information about any single individual cannot be deduced from the output.

**Epsilon Parameter ($\epsilon$)** The parameter $\epsilon$ controls the level of privacy guaranteed. A smaller value of $\epsilon$ implies greater privacy, as the algorithm's output is less influenced by any single record. However, an $\epsilon$ that is too small can reduce the utility of the data, while an $\epsilon$ that is too large can compromise privacy. Therefore, choosing $\epsilon$ is **crucial** for effectively balancing privacy protection with data utility.

## 2.3 Advantages of Differential Privacy

DP offers numerous advantages , making it a robust framework for analyzing sensitive personal information and protecting privacy. Among the main advantages we find:

- **Quantification of Privacy Loss**: DP precisely measures privacy loss for each mechanism, enabling comparison and balancing between privacy and data accuracy.
- **Composition**: By quantifying privacy loss, DP allows the management of overall privacy across multiple data operations, facilitating the design of complex algorithms from simpler components.
- **Group Privacy**: DP protects privacy for both individuals and groups, ensuring extended protection when handling collective information.

- **Closure under Post-Processing**: DP remains effective even after additional data processing, preventing analysts from diminishing privacy without extra information.
- **Choice of $\epsilon$**: DP offers precise control over the privacy-utility balance through the $\epsilon$ parameter, though selecting an appropriate value is crucial to maintain effective protection.

## 2.4 Example: Application of a DP Algorithm

Imagine being a data scientist specializing in social analysis, interested in studying a highly taboo behavior within a population. Each response in the dataset is a 'yes' or 'no', reflecting the truth of the interviewed individuals. Due to a strict privacy policy, the data holder does not allow **direct access** to the dataset.
As a Differential Privacy expert, you suggest to the data curator to use a DP algorithm to mask private information, thus allowing you to perform the analysis. The algorithm works in the following way for each individual response in the dataset:

1. **First Coin Toss:** A coin is tossed with a probability $p_{\text{head}}$ of landing heads.
   - **If heads:** The individual's original response ('yes' or 'no') is returned.
   - **If tails:** A second coin is tossed.
2. **Second Coin Toss (if tails on the first toss):**
   - **If heads:** 'Yes' is returned.
   - **If tails:** 'No' is returned.

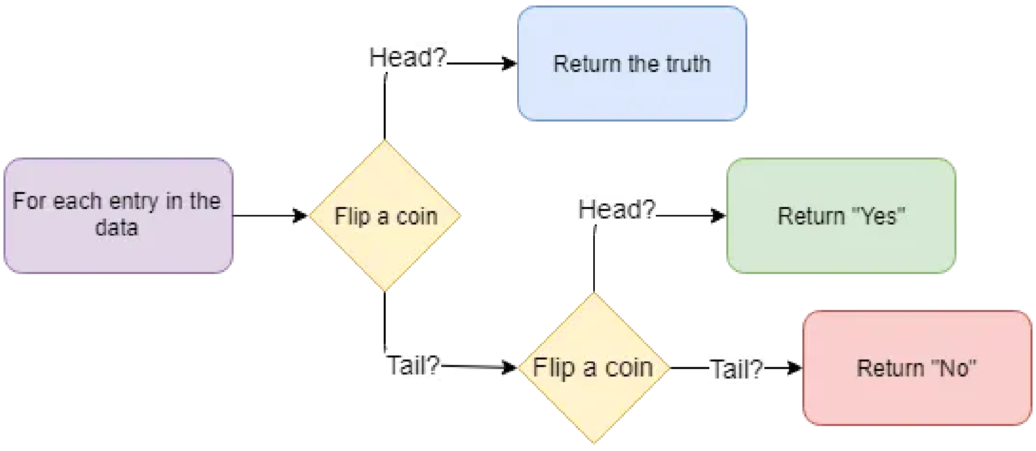

Figure 1: Example Scheme

**Conclusion:** This process introduces a controlled level of noise into the responses, masking each individual's specific information without significantly compromising the overall accuracy of the analysis. Consequently, even if statistical correlations emerge from the aggregated data, it is not possible to trace back to individual responses, thus ensuring the privacy of the respondents.

# 3 Project

## 3.1 Prerequisites

For the development of this project we chose the **Python** language, specifically version 3.11. It is because the frameworks and modules used are native to this language, and in the field of machine learning, it is the most widespread. In addition, a prior knowledge of machine learning is required for the formation of an artificial intelligence model.

### 3.1.1 Technologies Used

The following libraries were used:

- **PyTorch**: Build and train machine learning models with deep learning framework.
- **Torchvision**: PyTorch extension for predefined datasets, such as MNIST and pre-trained models.
- **Opacus**: Library to implement Differential Privacy in PyTorch models.
- **Matplotlib**: Library for creating graphs.

### 3.1.2 Dataset Selection

We chose *MNIST* dataset (Modified National Institute of Standards and Technology), one of the most famous datasets in the machine learning field for image classification. It consists of handwritten digit images in grayscale (28x28 pixels). The dataset contains:

- **10 classes (0 - 9)**
- **60,000** images as a training set
- **10,000** images as test set.

*PyTorch* and *Opacus* are optimized for image datasets and that's why an image dataset was chosen rather than a textual or tabular one.

**Note:** A subset of 10,000 samples was selected for training and 2,000 samples for testing, to optimize execution time and reduce computational resource consumption, still maintaining good model performance quality.

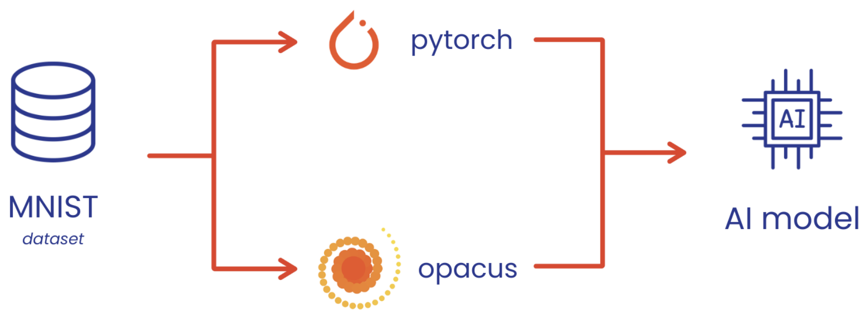

Figure 2: Flowchart

## 3.2 Project Structure

Project is hosted on GitHub and has the following structure:

```
Dataset-anonymizer               # Directory of the repository
    |
    |-- mnist                    # MNIST Dataset
    |-- src                      # All modules
    |-- docs                     # Report & presentation
    |-- requirements.txt         # Python modules to install
```

The project has been structured into separate modules to improve *readability, maintenance, and scalability.* The **src folder** is composed as follows:

```
   src
    |
    |-- plots                    # The resulting graphs are saved
    |-- model.py                 # Define and validate the ResNet model
    |-- data.py                  # Load and prepare the MNIST dataset
    |-- plotter.py               # Generate metric visualizations
    |-- trainer.py               # Train and evaluate the ML model
    |-- main.py                  # Orchestrate all process
```

## 3.3 Implementation

This section reports the most important code snippets related to the modules in *src.* You must install all the dependencies present in ***requirements.txt*** to be able to run the source code, then configure the parameters that you want present in `main.py`, such as *data_root, num_train_samples, batch_size*, and differential privacy settings (*epsilon, delta, clipping_threshold*).

### 3.3.1 data.py

Manages the loading and preparation of the MNIST dataset, including the reduction of the number of samples to optimize computational resources.

- load_data: is a function used to load and split the dataset
- get_dataloaders: allows obtaining the dataloaders related to the Train and Test data, specifically: A DataLoader is a tool that facilitates the loading and manipulation of data in batches for training machine learning models, offering options for data shuffling and parallel loading to improve the efficiency of the process.

```python
# Select a subset of the dataset based on the number of samples
def load_data(self, train=True):

    dataset = MNIST(root=self.data_root, train=train, download=True,
        transform=self.transform)
    if train:
        subset = Subset(dataset, list(range(self.num_train_samples)))
    else:
        subset = Subset(dataset, list(range(self.num_test_samples)))
    return subset

# Creates and returns DataLoaders for training and testing datasets.
def get_dataloaders(self):

    train_subset = self.load_data(train=True)
    test_subset = self.load_data(train=False)

    train_loader = DataLoader(
        train_subset,
        batch_size=self.batch_size,
        shuffle=True,
        num_workers=2,
    )

    test_loader = DataLoader(
        test_subset,
        batch_size=self.batch_size,
        shuffle=False,
        num_workers=2,
    )

    return train_loader, test_loader
```

### 3.3.2 model.py

It defines and validates the ResNet18 model, modifying the input to handle grayscale images and the 10 classes of the dataset. ResNet18 originally accept three-channel images (red, green, and blue). But we work with grayscale images (one color) so we adapt ResNet18 to take only one as input modifying the first convolutional layer to accept single-channel images.

```

class ModelBuilder:
    @staticmethod
    # Build a custom model specifically adapted to MNIST dataset
    def build_model():
        # Load ResNet18 configured for 10 output classes
        model = models.resnet18(num_classes=10)

        model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding
            =3, bias=False)
        model.fc =Conclusion nn.Linear(model.fc.in_features, 10)
        return ModelBuilder.validate_model(model)
```

### 3.3.3 trainer.py

Contains the classes and functions for training and evaluating the model, both with and without the implementation of Differential Privacy. To implement differential privacy, it was sufficient to follow the Opacus documentation.

```
# Enables DP for the specific model
def _enable_dp(self):
    self.privacy_engine = PrivacyEngine()
    self.model, self.optimizer, self.train_loader = self.privacy_engine.
        make_private_with_epsilon(
        module=self.model,
        optimizer=self.optimizer,
        data_loader=self.train_loader,
        epochs=self.epochs,
        target_epsilon=self.epsilon,
        target_delta=self.delta,
        max_grad_norm=self.clipping_threshold,
    )
    print(f"[DEBUG] Using sigma={self.optimizer.noise_multiplier} and C
        ={self.clipping_threshold}")
```

### 3.4 Results Obtained

We trained a total of four machine learning models using:

- **epsilon values of** [**1, 5, 10**] for 3 models
- One model without DP privacy leyer

Each model underwent **five training epochs**, allowing us to test the training process on a local machine. The result is aligned with expectations because the greater the parameter epsilon ( i.e. privacy minor ) the greater the model accuracy until maximum accuracy is obtained when no mask is applied to the data. It is very interesting to analyze how

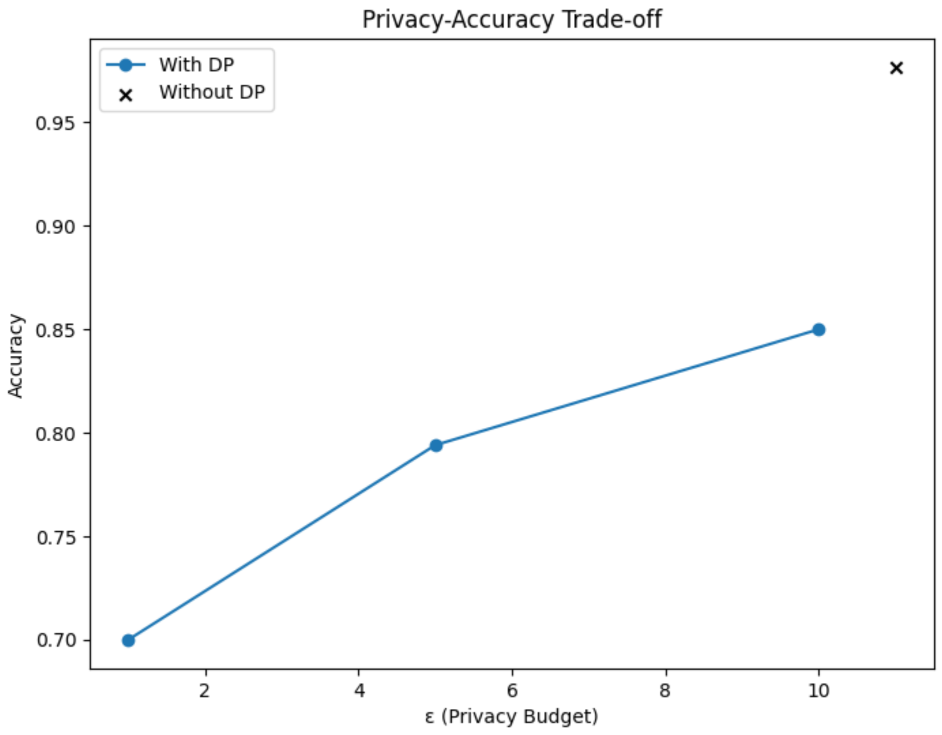

Figure 3: Privacy Accuracy tradeoff

the model, which is not pretrained, attempts to increase its accuracy with each epoch.

As expected, the results confirm that a model without Differential Privacy (DP) is easier to train and starts with higher initial accuracy. In contrast, models with smaller epsilon values begin with a lower accuracy and gradually improve to a slightly higher accuracy over time.

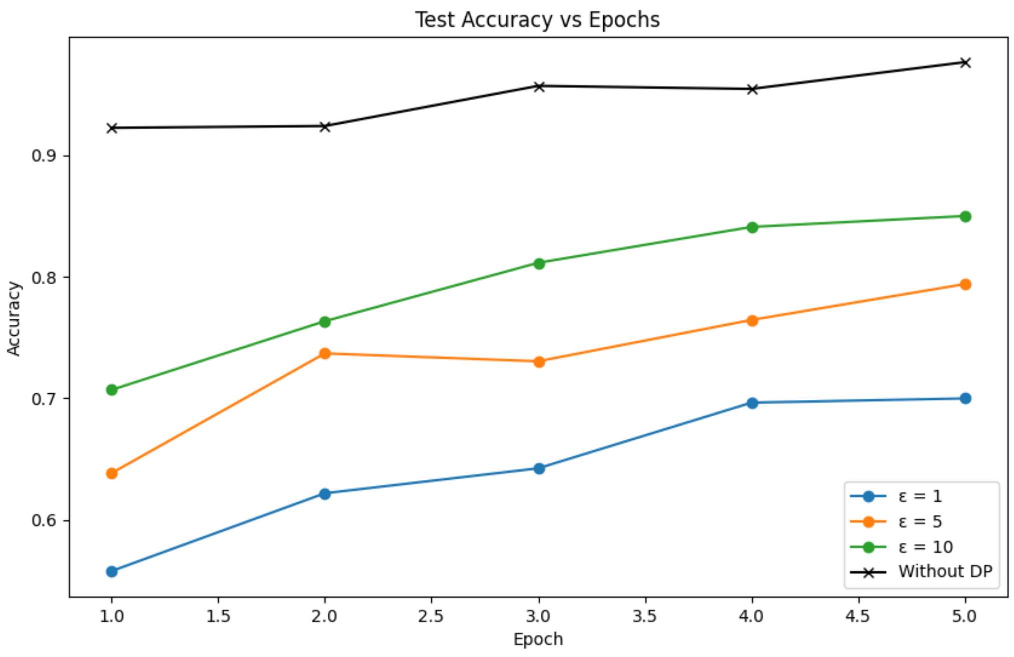

Figure 4: Accuracy vs Epochs

## 4 Conclusions

In this project, we successfully implemented a machine learning model (prototype) for data anonymization, using differential privacy. The use of the MNIST dataset, particularly a subset [10k,2k], allowed us to locally test the adopted approach, demonstrating how the integration of privacy mechanisms can influence the model's performance.
The obtained results clearly highlight the trade-off between privacy and accuracy: as the parameter $\epsilon$, which indicates lower privacy protection, increases, a significant improvement in the model's accuracy is observed. This confirms the theory that greater privacy protection can lead to reduced model performance. However, the accuracy achieved even with low $\epsilon$ values suggests that it is possible to effectively balance privacy and data utility.
During the development of the project, several challenges emerged. One of the main challenges was choosing the optimal value of $\epsilon$, which requires careful consideration of specific privacy needs and computational resource limitations. Additionally, adapting the ResNet18 model to handle grayscale images required modifications to the default model, highlighting the importance of customizing existing models for specific datasets.
The project's limitations include the use of a subset of the MNIST dataset, which may not fully reflect the complexities of larger and more diverse datasets. Furthermore, the limited number of training epochs imposed to optimize execution time may have affected the model's ability to fully converge.

## 5 References

1. `https://github.com/pytorch/opacus`
2. `https://privacytools.seas.harvard.edu/differential-privacy`
3. `https://towardsdatascience.com/understanding-differential-privacy`
4. `https://en.wikipedia.org/wiki/Differential_privacy`
5. `https://research.google/deep-learning-with-label-differential-privacy`