Project Report
Privacy Evaluation and Accuracy for Different ML Secure Models

Anastasiya Merkushova, Yannick Martin, and Alessandro Stanghellini

University of Basel
Privacy-Preserving Methods for Data Science and Distributed Systems
Spring Semester 2024

1 Introduction

Machine learning is increasingly being used in medicine. Examples include the diagnosis of diseases such as breast cancer and personalised medicine, where drugs are tailored to a patient's genome. These models are trained using particularly sensitive data. The responsible use of personal data is therefore particularly important in this area.

We have therefore implemented privacy mechanisms in the training of neural networks and analysed the impact of these mechanisms on accuracy and training time. The mechanisms we used were differential privacy with and without federated learning. To see if the privacy mechanisms really work, we tested the trained models with simple attacks and compared them with each other. The attacks we used were membership inference attacks and adversarial attacks.

Membership inference attacks aim to infer whether a data point has been used to train the model, which could be a sensible information.

Adversarial attacks expose vulnerabilities in machine learning models by using subtly altered inputs to deceive the models, highlighting potential security risks and the need for robust defences.

All our code and data to reconstruct our results can be found on our GitHub..

1.1 Dataset

The dataset is Texas-100: a hospital dataset was created starting from the Hospital Discharge Data records released by the Texas Department of State Health Services. The processed dataset contains 67,330 examples, each corresponding to a unique patient. Each record comprises 6,169 binary features, containing information about the patient, the causes of injury, the diagnosis, and the procedures the patient underwent. The data is clustered into 101 classes, representing the 101 most frequent medical procedures present. The dataset can be found here.

2 Model Creation

In this section, we review the methods and frameworks used to develop models with privacy. We focus on two main approaches: training neural networks with differential privacy and federated learning with local differential privacy. These techniques preserve the privacy of the data throughout the training process of the models.

2.1 Training Neural Networks with Differential Privacy

In the lecture, we saw that adversaries interacts with the model that has been trained with sensitive data, and thus extract training data or properties from the data. This is possible because the loss is different for training data and unseen data. More about possible attacks can be found in chapter 3: Membership Inference.

To protect our training data, we can use the differential privacy mechanism, namely the Differentially Private Stochastic Gradient Descent (DP-SGD) algorithm [5].

Algorithm 1 Differentially private SGD (Outline) [1]

Input: Examples $\{x_1, \ldots, x_N\}$, loss function $L(\theta) = \frac{1}{N} \sum_i L(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C
Initialize θ_0 randomly
for $t \in [T]$ **do**
    Take a random sample L_t with sampling probability $\frac{L}{N}$
    Compute gradient
    for each $i \in L_t$ **do**
        $g_t(x_i) \leftarrow \nabla_{\theta_t} L(\theta_t, x_i)$
    end for
    Clip gradient
    for each $i \in L_t$ **do**
        $\bar{g}_t(x_i) \leftarrow \frac{g_t(x_i)}{\max\left(1, \frac{\|g_t(x_i)\|_2}{C}\right)}$
    end for
    Add noise
    $\tilde{g}_t \leftarrow \frac{1}{L} \sum_i \bar{g}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 I)$
    Descent
    $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{g}_t$
end for
Output θ_T and compute the overall privacy cost (ϵ, δ) using a privacy accounting method.

The two most crucial aspects of DPSGD are the clipping of gradients in line 11 and the addition of Gaussian noise at line 14. The former is necessary to ensure that gradients have a defined upper bound so that we can determine the sensitivity which influences the magnitude of the added noise [5].

PyVacy To implement DPSGD, we wanted to use the Python library PyVacy. PyVacy describes itself as TensorFlow Privacy Library but for PyTorch. Unfortunately, the PyVacy Library is not well maintained. This means that the version you download with pip is not the current version on GitHub. This implies that the examples from GitHub do not work because the functions of the two versions have different parameters, or that certain classes cannot be imported at all. The problem is not really new, as it has existed since at least 2020, as there is an open issue on GitHub to this topic.

However, we somehow managed to get it running anyway.

Algorithm 2 Setting up PyVacy

```
1: Initialize DPSGD Optimizer:
2: optimizer = optim.DPSGD(
3:   params=network.parameters(),
4:   l2_norm_clip= C,
5:   noise_multiplier= ... ,
6:   batch_size= ...,
7:   lr= ...,
8: )
9: Compute Privacy Budget:
10: epsilon = analysis.moments_accountant(
11:   N=len(trainloader.dataset),
12:   batch_size=...,
13:   noise_multiplier=...,
14:   epochs=...,
15:   delta=...,
16: )
17: Now Start the training as usual
```

Only the optimizer needs to be defined as the DPSGD version (instead of Adam, for example). To define the optimizer, a number of parameters are required, including a noise multiplier and the clipping parameter C (l2_norm_clip). In addition, there is the moments_accountant function to calculate the total privacy budget. The privacy budget depends on the noise multiplier, batch size, number of epochs, delta and number of data entries. Unfortunately, there is no option to define a target privacy budget for the function instead of the noise multiplier. Because of this and also because Opacus is much better maintained, we changed the library for the analysis.

Opacus Opacus is also a Python library which implements DPSGD for PyTorch. Like PyVacy, Opacus requires also only minimal changes to the code to implement DPSGD.

Algorithm 3 Setting up Opacus

```
define your components(network, optimizer) as usual
Initialize Privacy Engine:
privacy_engine = PrivacyEngine()

Make Network Private with Epsilon:
network, optimizer, trainloader =
  privacy_engine.make_private_with_epsilon(
    module=network,
    optimizer=optimizer,
    data_loader=trainloader,
    max_grad_norm=C,
    target_epsilon=epsilon,
    target_delta=Delta,
    epochs=epochs
)
Now Start the training as usual
```

The big advantage with the function make_private_with_epsilon is that we can define a target_epsilon (privacy budget), and the function adjusts noise so that this privacy budget is not exceeded. This privacy tracking is calculated by privacy accounting. Rényi differential privacy accounting is used for this purpose. However, there is also the option of using your own privacy accountants via an interface[4].

Network Our neural network consists of three linear layers. The first and second layer use the hyperbolic tangent as an activation function, while the third layer outputs the final predictions. Input size is 6169, and it outputs 101 classes with one hidden layers of size 128. We have tried out various network architectures, but did not include them in the analysis. Because with a single model, there are already enough hyperparameters to optimize. However, we achieved better results with the classic activation function tanh than with more modern ones such as ReLu when using DPSGD.

2.2 Federated Learning with Local Differential Privacy

Federated learning is used when data is distributed across different parties. With our data set, this could be different hospitals from different cantons. However, the different parties (hospitals) do not want to send the data anywhere so that

a model can be trained. The solution is federated learning. The parties train the neural networks locally with a predefined model. The trained models are then sent to a central server (aggregation module), where the models are combined to form a global model (as visible in Figure 1).

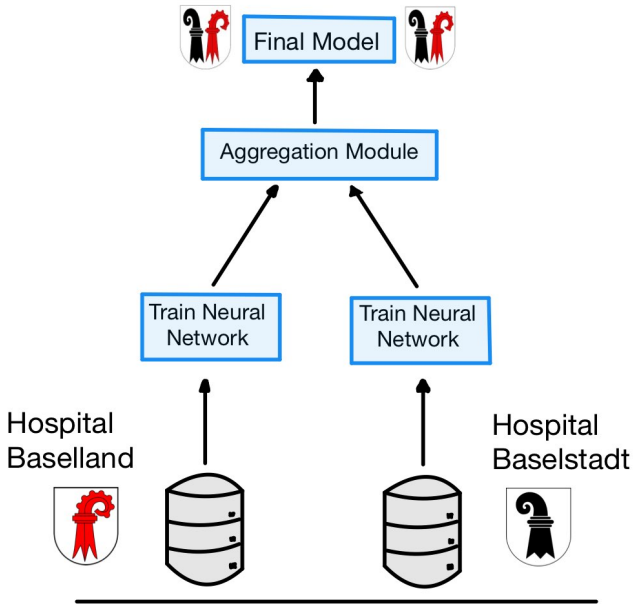


Fig. 1: Federated Learning Architecture

The advantage is that we don't have to send the sensitive data, but still, the server could learn something about the sensitive training data through the model output, just like in normal training. To prevent this, the neural networks are trained locally with DP-SGD. This is known as federated learning with local differential privacy [5].

We wanted to use the python library PySyft to implement this. There are many examples of FL with PySyft, but they are all based on hookers, which are no longer available in the new PySyft version. There are FL tutorials on GitHub, but only for one client (they call it Worker because the data and model are sent from the server to the client). But we have not been able to connect multiple clients to the server. Since PySyft also does not have a function to aggregate models, we simply simulated FL without a library. In other words, without sending the models. To aggregate the models, we simply used parameter averaging. Again, there would be better methods to do this. We have searched the internet but have not found a library with which FL could be easily implemented, especially not with differential privacy.

2.3 Results

Normal Learning - Hyper-parameter finding SGD

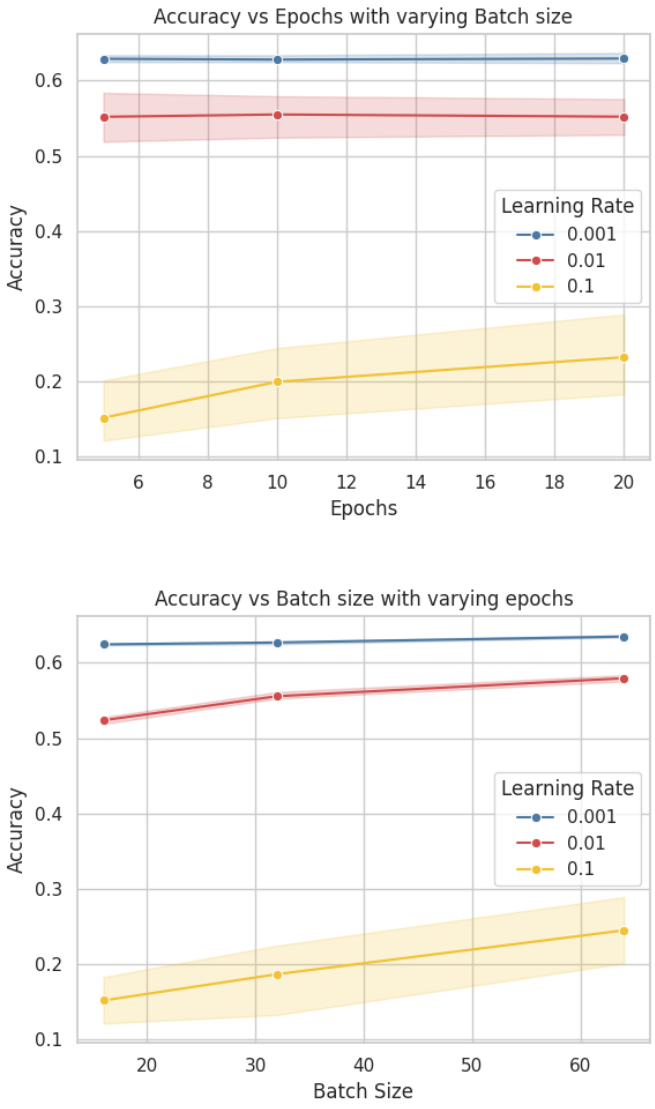


Fig. 2: Accuracy Plots to find Hyper-parameters for SGD

To find the good hyperparameters for our model, we trained the model with different hyperparameters and then displayed the results in two accuracy line pots Figure 2. On both plots, you can see that a learning rate of 0.001 gives the best accuracy. If the learning rate is so small, the number of epochs and batch size no longer have much influence when these are increased. A learning rate of 0.1 results in much worse accuracy. However, you can see for this learning rate that an increase in the epochs and an increase in the batch size lead to better accuracy. Based on this analysis, the learning rate must be 0.001. We would set the number of epochs to 10 and the batch size to 64, so that high accuracy is achieved.

Hyperparameter finding DPSGD

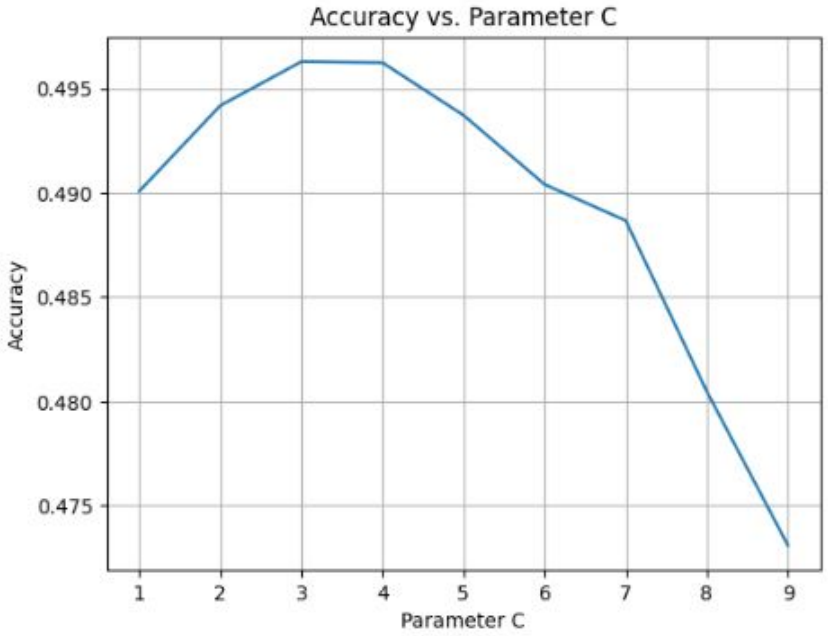


Fig. 3: Determine Clipping Norm

To find the hyperparameters for DPSGD, we need to determine the clipping norm parameter C for fixed parameters. As mentioned in the lecture, we get the best accuracy when the clipping norm is 3 or 4, which can also be seen on the plot (fig. 3). For all the following plots, we have decided to set the clipping norm to 3.

Now we have done the same as for the SGD hyperparameter finding. We calculated the accuracy for different batch sizes, number of epochs and learning rate. These plots (fig. 4) show the results for epsilon = 1. The plots for other

epsilon values can be found in the appendix. Similar to SGD, the learning rate is very important for the accuracy. Again, a learning rate of 0.001 gives the best results. Another difference is that a higher number of epochs is required to achieve the best accuracy. Varying batch sizes also have a greater influence here than with SGD. To achieve the best accuracy we would set the parameters as follows: Lr = 0.001, batch size = 64 and epochs = 20.

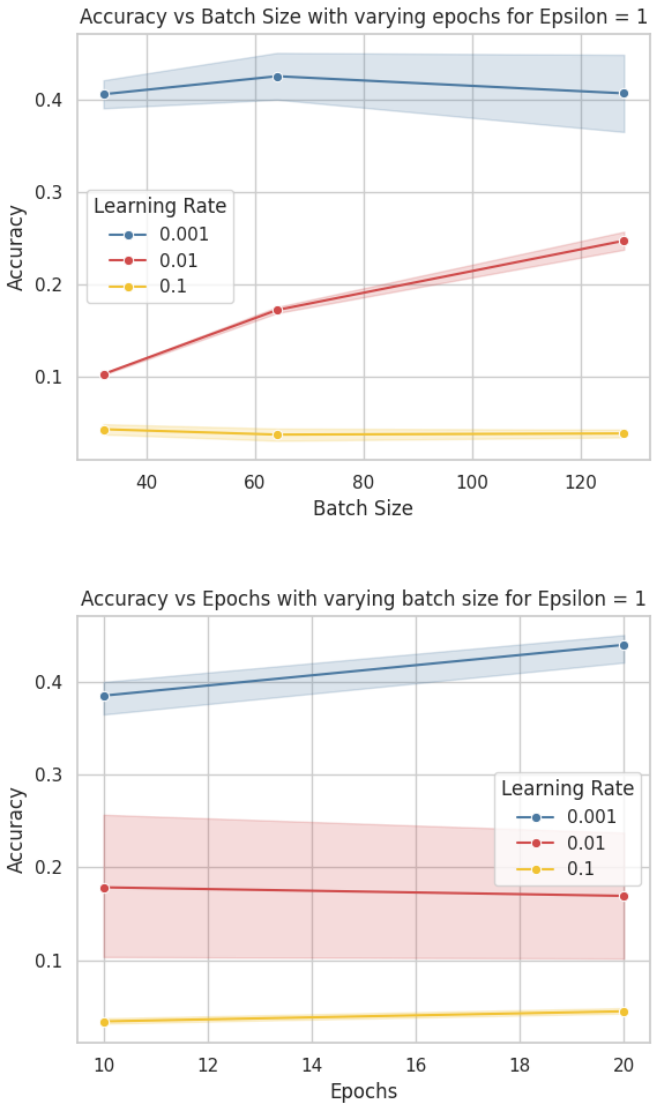


Fig. 4: Accuracy Plots to find Hyper-parameters for DPSGD and epsilon = 1

Final Analysis

The best models are listed in this table 1 below. Three models were created with PyVacy. However, the PyVacy models are significantly worse than the models created with Opacus. This means that Opacus is not only easier to use, but also delivers better results (at least for our network and dataset). The best accuracy we were able to achieve was 63%, which does not seem very high at first sight, but you have to bear in mind that a random classifier with 100 classes would have an accuracy of 1%, so with this knowledge, 63% is not so bad.

Model	Accuracy	Epsilon	Epochs
NN	0.63	0	10
Opacus	0.55	20	20
Opacus	0.53	10	20
Opacus	0.45	1	20
PyVacy	0.37	28	20
PyVacy	0.38	12	20
PyVacy	0.26	1.3	20

Table 1: Comparison of different models with differential privacy

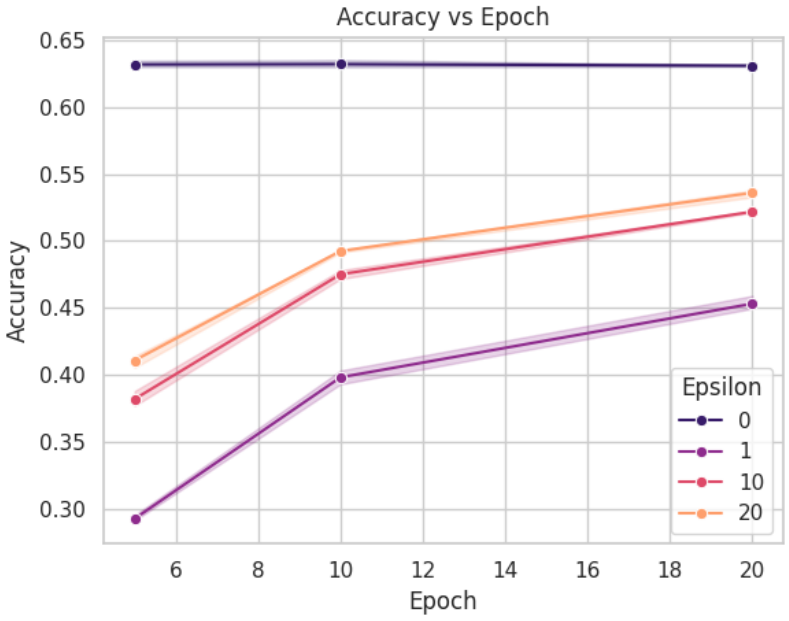


Fig. 5: Accuracy vs Epoch line plot

For the following analysis, the learning rate was then set to 0.001 and the batch size to 64. The precision line plot fig. 5 now shows what was previously visible. With the privacy preserving mechanism, the Accuracy is worse. The smaller the Epsilon, the lower the accuracy. An epsilon of 10 and 20 have a very similar level of accuracy, while an epsilon of 1 has an accuracy that is 10% worse. What you can see here is: For DP SGD more iterations (epochs) are needed until the result converges. For DPSGD it is a little more than 20 epochs till they reach the best accuracy. Without DPSGD it converges after 5 epochs. As a conclusion, we can say that for epsilon = 10 we get an accuracy of 53%. This is still quite good and guarantees a certain privacy.
We have created many other plots for this analysis, including box plots. They all show the effects described above. The plots are included in the appendix.

In terms of training time, we can say that DPSGD takes about 2.5 times longer than SGD. However, there is also a very slight trend that more epsilon leads to a longer training time for the same number of iterations. But this effect is not very strong. .

It should also be noted that without privacy the optimum is reached after 5 iterations, while with privacy preserving training it takes at least 20 iterations. This means that the privacy preserving implementation is about 10 times slower than training without privacy. All of this this can be seen in this bar chart below fig. 6.

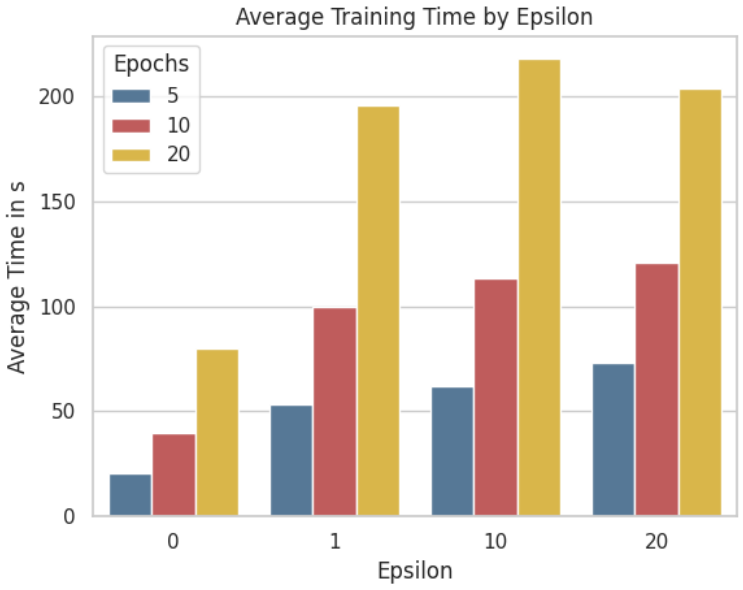


Fig. 6: Training Time on GPU for different models and epochs

Federated Learning

Since we are using the same network and the same data for the federated learning, we have directly determined the learning rate, the clipping bound of the batch size C, and have kept it constant (lr = 0.001, batch size = 64, C = 3).

The plot (fig. 7) below shows the accuracy for the trained client, i.e. before the models were aggregated, and the accuracy for the aggregated models. Both accuracies were calculated using each client's private test set. The result is that regardless of the number of iterations, the accuracy for the training client is better than for the aggregated model. The same result is also visible for models with DP. This is obviously not a desirable result. Heterogeneous data distribution or a poor aggregation method could be possible reasons for this. However, our assumption is that the data distribution is homogeneous, so it may be that the aggregation procedure is not the best. This is something we would need to look into further.

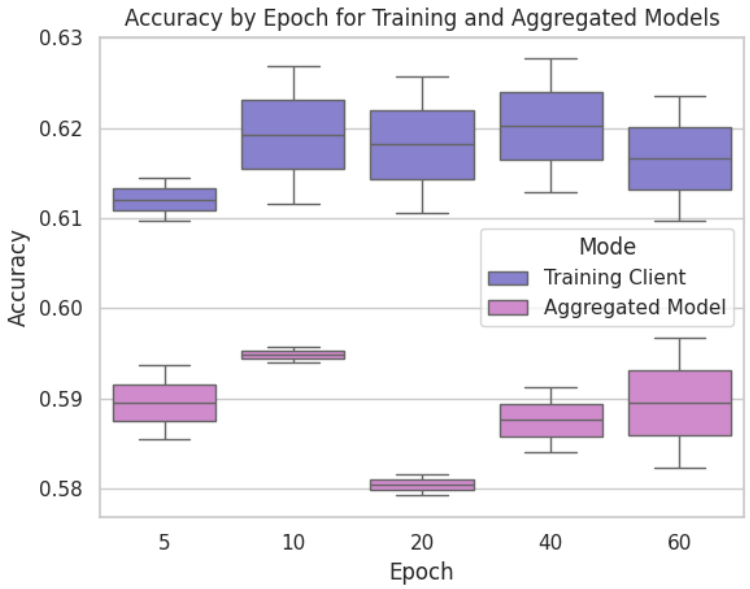


Fig. 7: Accuracy of trained clients and aggregated models

If we compare all the trained aggregated models like in the plot below fig. 8, we see similar results as with normal training. The accuracy without privacy mechanism is slightly below 60%, but also converges after 5 iterations. The Opacus models are all slightly worse: the lower the privacy budget, the worse the accuracy. However, it takes about 40 iterations to reach the best accuracy.

It must also be said that the FL models are worse than the normal models. For example, for epsilon = 10 , the FL model is 10% worse.

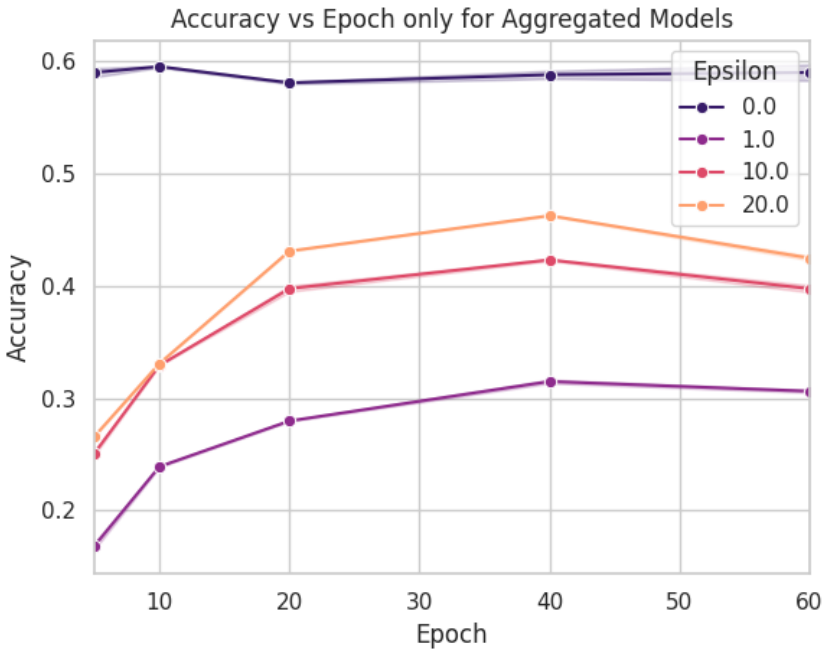


Fig. 8: Accuracy line plot for different FL models

The distribution of the training time fig. 9 of the clients looks very similar to the one without FL. Here the training time is of course faster, because each client has less data. But it takes more iterations to reach the optimum, so the actual training time is very close to normal training. There is also a slight tendency for training time to be slightly longer as the epsilon increases. In general, it can be said that the FL without privacy is a little more than 10 times faster than the one with privacy.

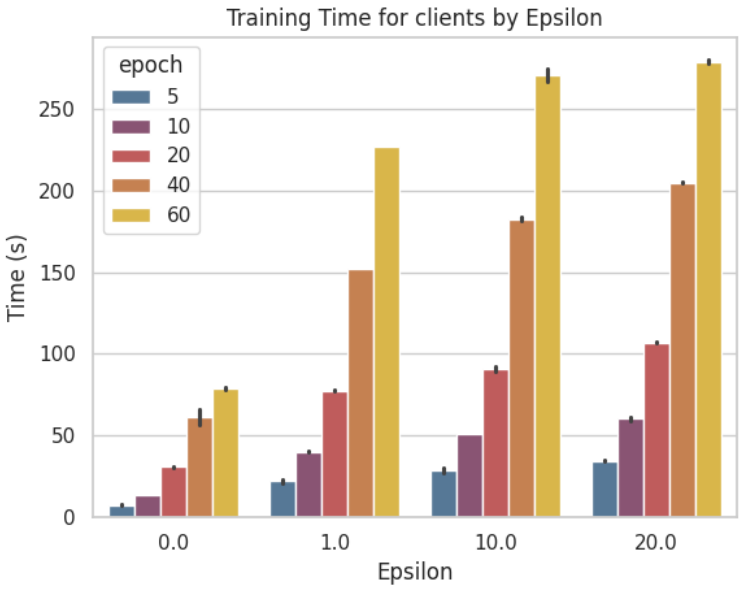


Fig. 9: Training Time on GPU for the Clients

2.4 Conclusion

Although differential privacy offers significant benefits in terms of protecting individual privacy, these benefits do not come for free. Typically, training differential privacy neural networks means accepting a loss of accuracy and a longer execution time. In addition, it is sometimes difficult to reproduce older examples, as many newer library versions are incompatible with other libraries. Additionally, the fact that there is no Python library that combines federated learning for pytorch and differential privacy in a common library makes it more difficult to implement such systems. These limitations highlight the need for further work in this area to balance privacy, accuracy and usability.

3 Membership inference attacks

3.1 Concepts

With machine learning models being applied in many different scenarios, from classification, text generation, multimedia recognition and many more, the problem of how secure from a privacy point of view these model are has become more important. Many recent studies have shown that one of the possible leakage of sensible information comes from membership inference attacks. These attacks aim to determine whether a sample of data was part of the training set for the model. Maybe at first one could think that this would not pose a significant threat to private information from the attacker, but we need to remember that many machine learning models are trained on sensible data, like preferences, purchases, health information, locations and so on.

Consider for example a model trained to detect some disease from medical images: if the attackers can infer which images are used for training, with the additional information of knowing the owners of those images, they can assume that the owners are actually effected by the disease.

One of the main reasons why membership inference attacks are potentially effective is that often machine learning models are over-parameterized and overfitted and this allows the model to store more information than those required to only fulfil its purposes, which means that the model learns something more about the training datasets. This happens also because of the high number of epochs over which the model is trained. In practice, this means that for training data the model behaves differently and this is observable in the model's parameters. In particular, as one can expect, the model will reach, for example, higher classification accuracy on training data records.

In literature, two main groups of membership inference attacks are defined: black-box and white-box attacks. This distinction is based on the adversarial knowledge, which is the amount of information that the attacker has or can receive about the model.

- In black-box attacks, the adversary only has access to the model's outputs, which could be the predicted labels or confidence scores (probabilities) for given inputs. The adversary does not have access to the model's internal parameters or structure.
- In white-box attacks, the adversary has full access to the model's internal parameters, architecture, and training process. This access allows the attacker to exploit detailed information about how the model was trained and how it functions internally.

On top of that, black-box attacks can be further differentiated depending on the information provided by the prediction output. This could include only the prediction labels, the top-K confidence scores or the full set of confidence scores.[2]

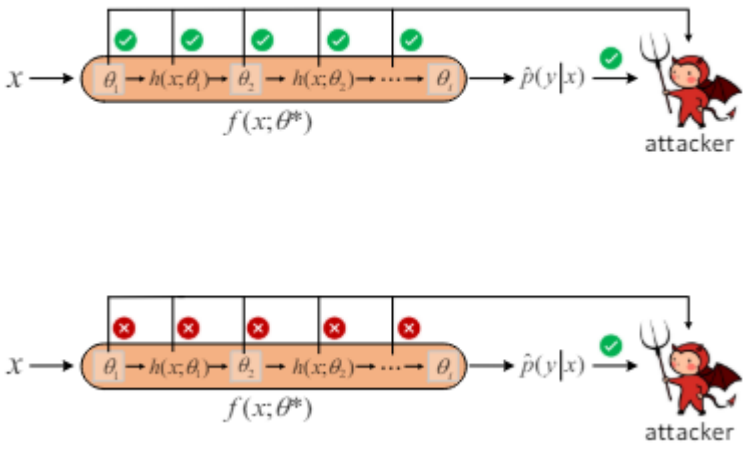


Fig. 10: Overview of white-box membership inference attacks on top and black-box at the bottom.[2]

In the picture x are the inputs, y the labels or outputs with posterior distribution $\hat{p}(\cdot\,|x)$, f represents the model depending on the input x and the optimized parameters θ^*. More in detail, the model is divided in hidden layers h, depending again on intermediate parameters θ_i.

3.2 Implementation and Results

Following the argument introduced in the previous paragraph on why membership inference attacks can be effective, our first idea was to build two simple (without additional libraries) attacks to the models trained, with and without privacy, and confront the results.

More in detail, our first approach consists in the following steps:

- build a sample of the whole dataset with the same proportions between train ant test records as the what's used to train the models;
- load the models (with and without differential privacy);
- evaluate the model on the sample points and calculate the loss with respect to the true labels;
- select the inferred members as the ones with smaller loss.

For the last step, the choice can be performed by adding the extra assumption that the attacker knew the amount of members in the sample dataset or by using a threshold level for the loss measurement.

Notice that our black-box attack relies on having the full confidence scores as output information. Thanks to this, we can calculate the prediction loss.

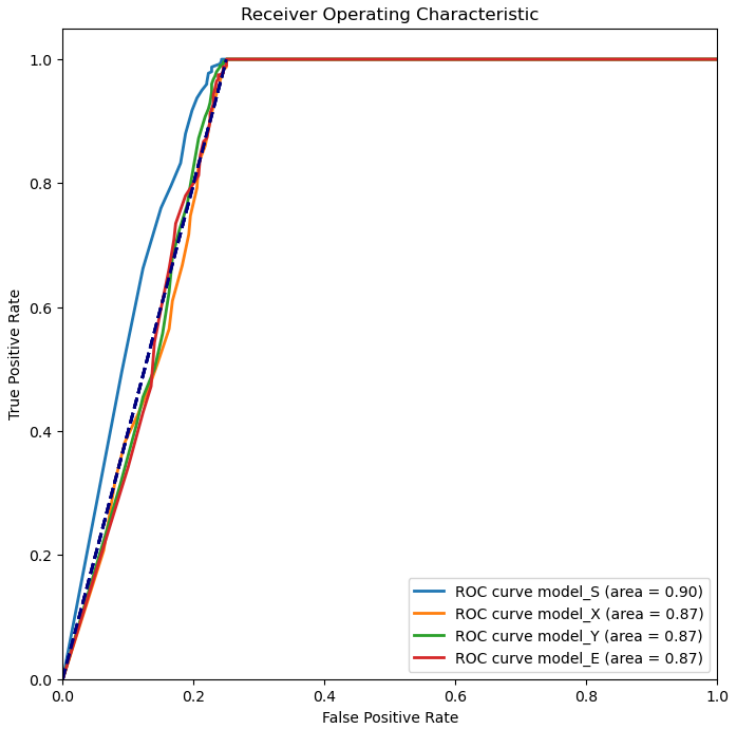
With the additional assumption of knowing the total amount of unknown members in the sample dataset, this attack gives the following results:

	Model S	Model X	Model Y	Model E
# Predicted members	400	400	400	400
# Correct predictions	329	318	321	321
Accuracy	82.25%	79.5%	80.25%	80.25%

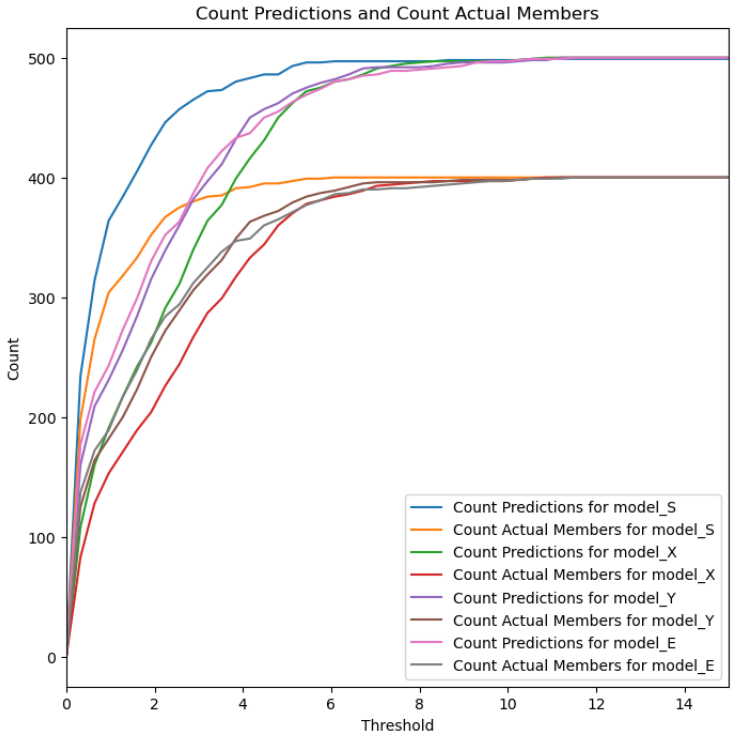
Before jumping to conclusions, it's worth to remember that our model is trained on 80% of the dataset, thus randomly picking a record would give us a training point with probability equal to 0.8. We notice that the only case in which the attack is reasonably better than random is the non-private one. Also, the worst of the other three cases is the one with the highest levels of privacy.

Another thing that must be considered for these results is that all these models are not extremely accurate, and that's why it's harder to see large differences between for the accuracy of the attack to different models.

Without the extra assumption just mentioned, we can analyze the predictions of training members using a threshold t, whose span is the range of the loss values across the sample dataset. By calculating the true positive rate and the false positive rate for each threshold, we can plot the ROC curve:



We can clearly see from the curves that the attack is more efficient on the non-private model, but as before this difference is clear but not huge for the same reasons mentioned previously. Also notice how from this graph, the three private models appear to behave very similarly. However, if we look at the net count of the predicted (correctly and not) members, we notice that the growth for model X (green and red lines) is sensibly slower than the other two (pink-grey and purple-brown lines). In the non-private model instead, it's way faster to correctly infer membership.



On the other hand, the second approach is a white-box attack. The idea is very similar to the previous one, but instead of computing the loss for each point, we measure the norm of the gradient of the loss function with respect to the input evaluated in each point of the sample dataset. The idea behind this attack is that the data points used to train the model should already fit the training parameters reasonably well and this means that variations of the loss function with respect to the input should be small for members.

The attack is white-box because in order to perform those gradients we need to know the inner parameters of the model and to have access to hidden layers.

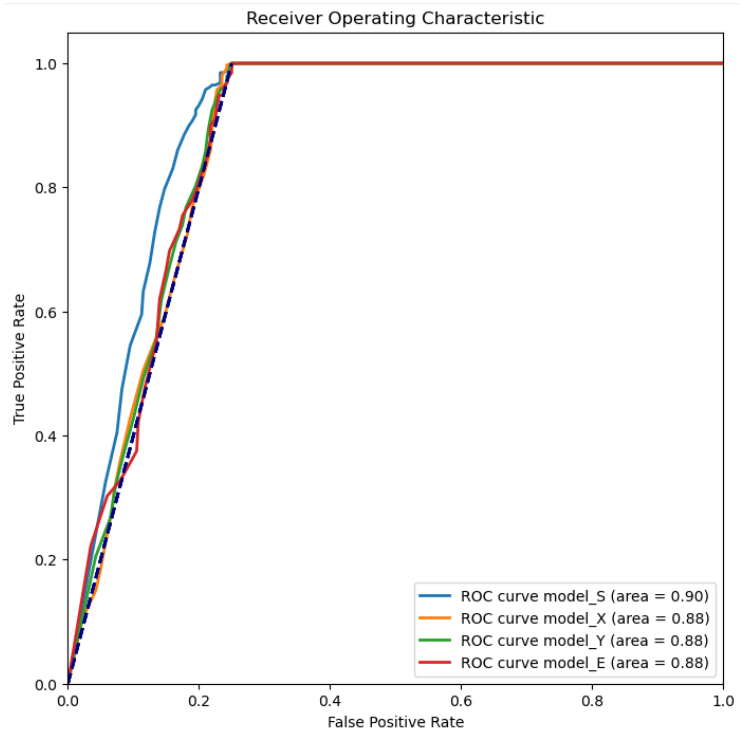
Similarly as before, we can split the process in the following steps:

- build a sample of the whole dataset with the same proportions between train ant test records as the what's used to train the models;
- load the models (with and without differential privacy);
- calculate the Euclidean norm of the gradient of the loss with respect to the inputs evaluated in the sample data points;
- select the inferred members as the ones with smaller gradient norm.

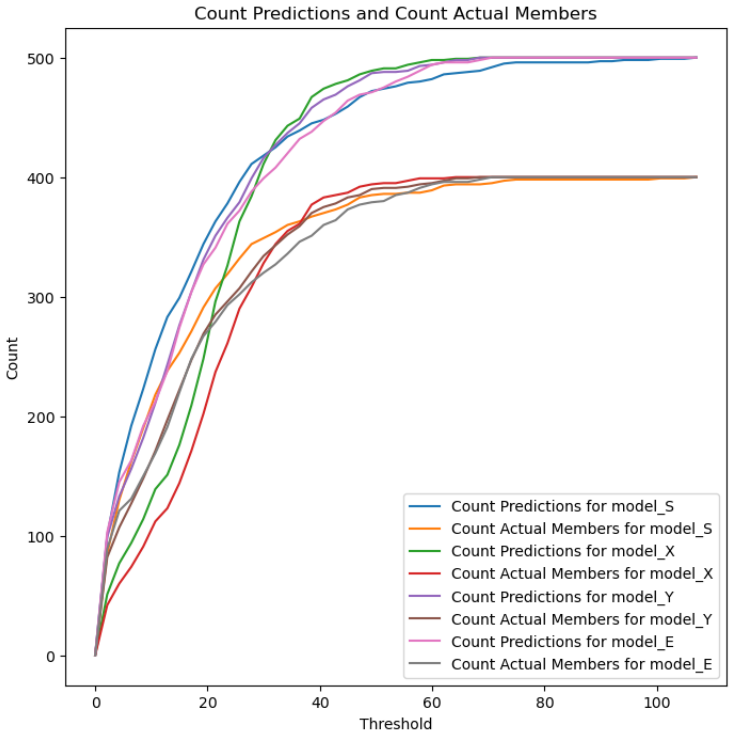
Again with the same extra assumption as before, we have the following results:

	Model S	Model X	Model Y	Model E
# Predicted members	400	400	400	400
# Correct predictions	336	318	322	321
Accuracy	84%	79.5%	80.5%	80.25%

As before, if we drop this assumption we can perform an analysis where the choice of the inferred members is based on a certain threshold.



Same script as for the previous attack: the efficiency is slightly better than the previous one for the non-private model, and the AUC (Area Under the Curve) is the same for every version of the private models. Again as before, if we look at the net count of the predicted members, the growth for model X (green and red lines) is sensibly slower than the other two (pink-grey and purple-brown lines).



In conclusion, we could simply say that adding privacy to a model in the form described before is helpful against membership inference attacks. In our specific case, any level of privacy added to the model makes the attack as accurate as a random pick, which is good but could be the consequence of the low accuracy that our private models have.

3.3 Privacy Meter

The ML Privacy Meter is a tool which analyses how prone a machine learning model is to membership inference attacks. The tool generates attacks on a trained target model assuming black-box or white-box access to the model to get the inference accuracy of the attack. White-box attacks can exploit the target model parameter's gradients, intermediate layer outputs or prediction of the model to infer training set membership of the input, while black-box attacks only use the target model predictions to identify membership [3].

Background

Privacy Meter a framework that enables auditing the privacy loss of a machine learning models about *a particular record*, in the black-box setting (where only model outputs — and not their parameters or internal computations — are observable). This framework has three main components:

1. Inference Game
2. Indistinguishability Metric: Measures the privacy risk.
3. Membership Inference Attack (MIA): Treated as hypothesis testing.

Depends on the definition of Inference Game, we can get different attacks. For instance, they have a Membership inference game for average model and record, where both the target model and the target record are randomly generated. And they compute the performance of the attack, by averaging it over many repetitions of this random experiment. However, this also limits the type of leakage that this game captures, as it is averaged over multiple target models and data records. On the contrary, there is another model - Membership inference game for a fixed model. This game is similar to the game for average model and record, except that the challenger always selects the same target dataset and target model across multiple trials of the game. Therefore, it quantifies the privacy loss of a specific model trained on a fixed dataset. Similar games are widely used in practical MIA evaluations for auditing the privacy loss of a released model in machine-learning-as-a-service setting [6].

For the metric, they use an indistinguishability measure to define privacy of individual training data of a model. According to this measure, the privacy loss of the model with respect to its training data is the adversary's success in distinguishing between the two possibilities 0 vs. 1 over multiple repetitions of the inference game. Naturally, the inference attack is a hypothesis test, and the adversary's error is composed of the false positive (i.e., inferring a non-member as member) and false negative of the test.

Specific Attacks and Optimisation

General attack strategies may not be effective for specific target models or data records. Hence, they design model-dependent and record-dependent thresholds for attacks, optimising them for specific targets. This approach improves attack performance by reducing the attacker's uncertainty about the target model.

They modelled several types of attacks:

- Via Shadow Models
- Via Population Data (Model P)
- Via Reference Models
- Via Distillation

Model-Dependent Attack P

To reduce computation we chose to work with the model that doesn't require training, neither shadows models nor reference once. Attack via Population Data relies on comparing the model's loss for the target data point against a threshold derived from a population dataset. Attack P uses different thresholds for different target models, but still uses the same threshold for different target data. And it uses following steps:

- *Setup:*
  We have a target model θ trained on a dataset D and a specific data point z (with features x_z and label y_z).
- *Loss Calculation:*
  Calculate the loss $\ell(\theta, x_z, y_z)$, which measures how well the model predicts y_z given x_z.
- *Population Data:*
  Use a general dataset (population data) to estimate the distribution of loss values for non-member data points on the target model θ. This population data is independent of the model's training data.
- *Threshold Determination:*
  Calculate the loss $\ell(\theta, x)$ for each data point x in the population data. Determine the α-percentile of these loss values to set the threshold $c_\alpha(\theta)$. This ensures that α the fraction of non-member instances in the population data have loss values below this threshold.
- *Hypothesis Test:*
  Compare the calculated loss $\ell(\theta, x_z, y_z)$ to the threshold:

$$c_\alpha(\theta): \qquad \text{if} \qquad \ell(\theta, x_z, y_z) \leq c_\alpha(\theta) \tag{1}$$

  conclude that z was likely in the training dataset D. Otherwise, conclude that z was not in the training dataset.

The threshold $c_\alpha(\theta)$ is model-dependent but does not depend on the specific data point z. It is derived from the population data to ensure a consistent false positive rate.

Reports

The framework provides results in the form of audit report, including:

- ROC curve

  A graph illustrating the performance of a classification model at various decision thresholds. The AUC (Area Under the Curve), represented in blue, is a threshold-independent measure of the classifier performance. A higher AUC is an indicator of a system vulnerable to the chosen metric. For reference, a random classifier yields an AUC of 0.5, while a perfect classifier yields an AUC of 1.0
- Confusion Matrix

  A graph illustrating the performance of a classification model for a specific decision threshold. Higher values on the top-left to bottom-right diagonal is an indicator of a system vulnerable to the chosen metric, while higher values on the top-right to bottom-left diagonal is an indicator of a system less vulnerable to the chosen metric.
- Signal histogram

  A histogram of the signal used by the chosen metric, on both members and non-member samples. A clear separation between the two groups is an indicator of a system vulnerable to the chosen metric.
- Vulnerable points Points that are most vulnerable to the chosen metric. The score depends on the chosen metric, but is always between 0 and 1, with 0 meaning low vulnerability and 1 high vulnerability.

However, for clarity, we included only ROC curves and signal histograms.

Results

The framework also includes the ability to train models and subsequently assess their vulnerability to attacks. For our experiments, we utilized the following configuration: model type: nn, 20 epochs, Adam optimizer, batch size of 64, and a learning rate of 0.01. We used the Texas 100 dataset, the same dataset as in previous studies [3]. However, we couldn't directly compare our results with those from the referenced paper due to the absence of Texas 100 dataset results there.

In our experiments, the model achieved an accuracy of 0.59 on the test dataset and 0.76 on the training dataset. Additionally, using their framework, we evaluated the vulnerability of this model to attack P (as described earlier) and obtained the following results (in black-box settings):

Also, we set up MIA via Population Metric on each of our model trained before and got these results:

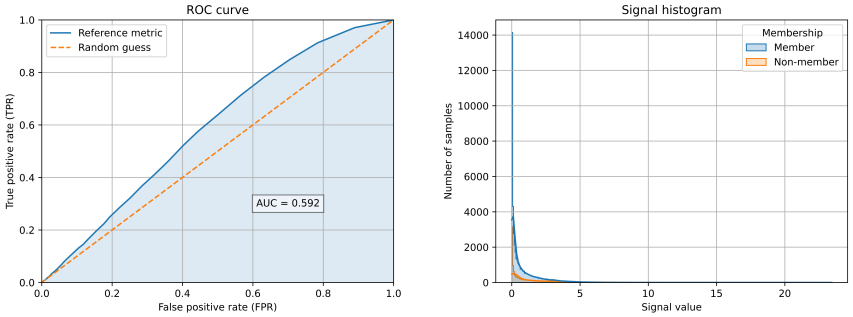


Fig. 11: Privacy Meter results for model created using their framework

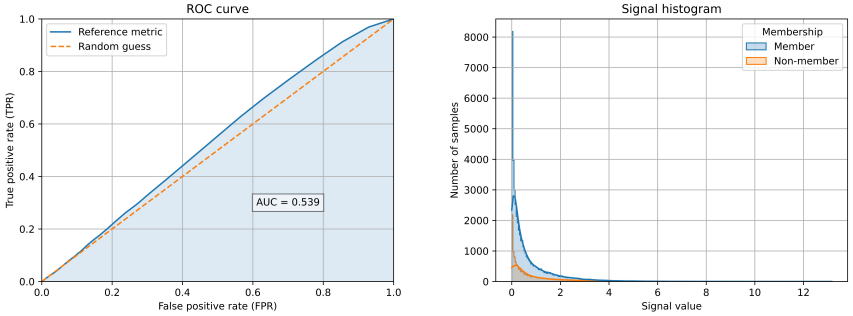


Fig. 12: Privacy Meter results for the Model S

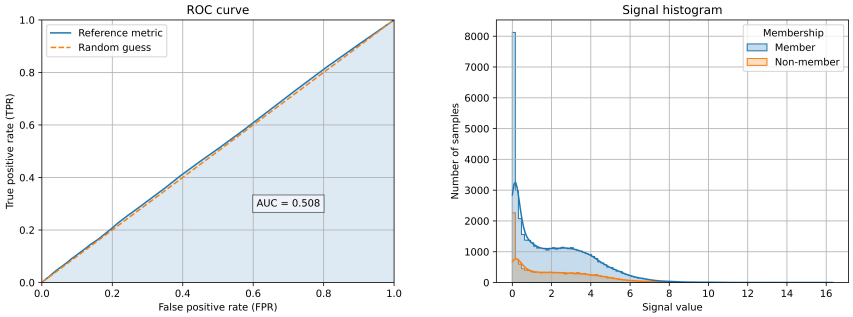


Fig. 13: Privacy Meter results for the Model Y

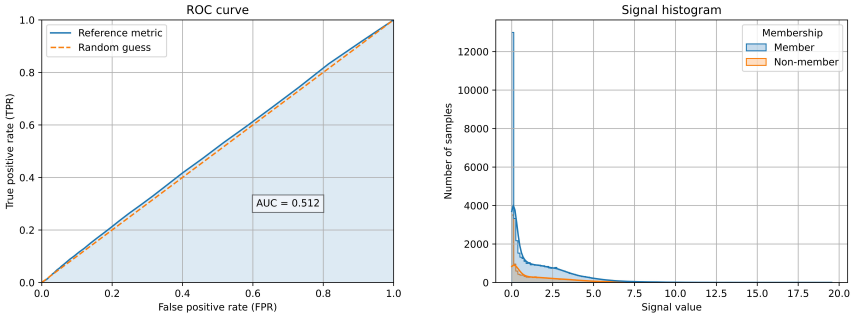


Fig. 15: Privacy Meter results for the Model E

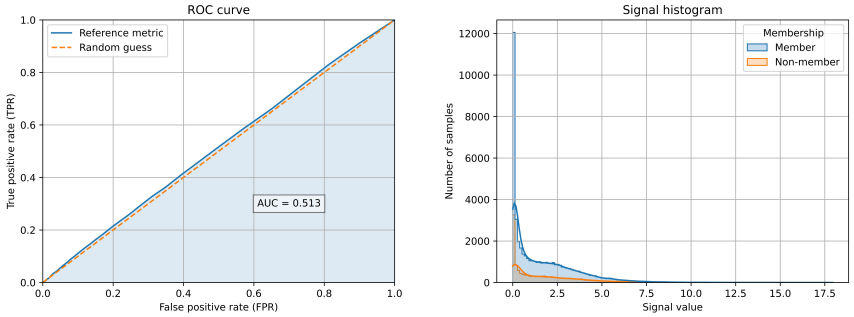


Fig. 14: Privacy Meter results for the Model X

In conclusion, our results confirmed that adding privacy to a model is helpful. We were able to distinguish between private and non-private setups. Interestingly, even without additional privacy methods, our model demonstrated a high level of privacy, although it did not achieve high accuracy. This situation resembles overfitting, where the test accuracy remains consistently below 0.6 across most setups, while the training accuracy occasionally reaches 0.85. Also, determining an accurate threshold for MIA depends on population data, which is influenced by the model's training data.

In addition, we noticed that people faced problems with Privacy Meter, while testing it with other dataset, that raise some questions about relatability of their attacks. This issue needs further investigation.

4 Conclusion

Throughout this project, we developed and trained a neural network model for medical dataset classification using various privacy-preserving techniques, achieving accuracy comparable to published results. Additionally, we conducted a Membership Inference Attack using two distinct methodologies, one of it using a state-of-the-art framework.

Looking ahead, our future work entails implementing multiple models to enhance accuracy on the current dataset and incorporating diverse datasets, such as medical images, to broaden our analysis. Furthermore, we plan to explore additional types of attacks, including adversary attacks, to bolster the robustness and security of our model.

References

1. M. Abadi, A. Chu, I. Goodfellow, et al., "Deep Learning with Differential Privacy", 2016.
2. C. Hu, Y. Tang, J. Wang, et al., "Membership Inference Attacks on Machine Learning: A Survey", 2021.
3. S. Kumar and R. Shokri, "ML Privacy Meter: Aiding Regulatory Compliance by Quantifying the Privacy Risks of Machine Learning", The 20th Privacy Enhancing Technologies Symposium, Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs), 2020.
4. A. Yousefpour, et al., "Opacus: User-Friendly Differential Privacy Library in PyTorch", 2022.
5. I. Wagner, Lecture Slides of "Privacy-Preserving Methods for Data Science and Distributed Systems", 2024.
6. L. Watson, C. Guo, G. Cormode, A. Sablayrolles, On the Importance of Difficulty Calibration in Membership Inference Attacks, 2022.

A Appendix

A.1 Additional Results

Hyperparameter analysis NN with Privacy

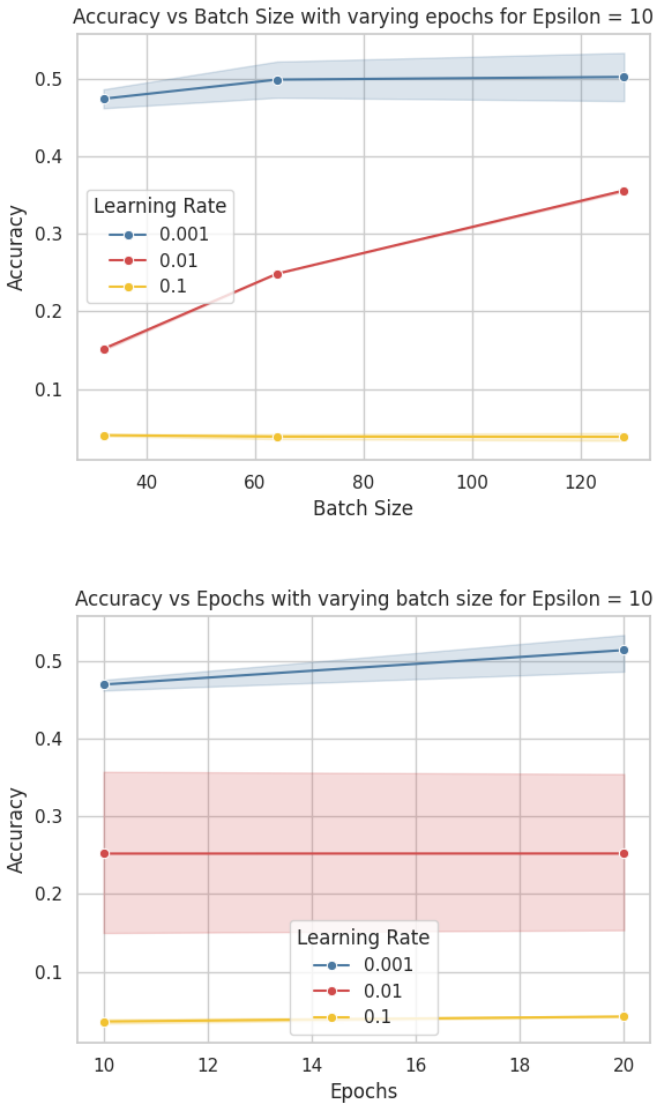


Fig. 16: Accuracy Plots to find Hyper-parameters for DPSGD and epsilon = 10

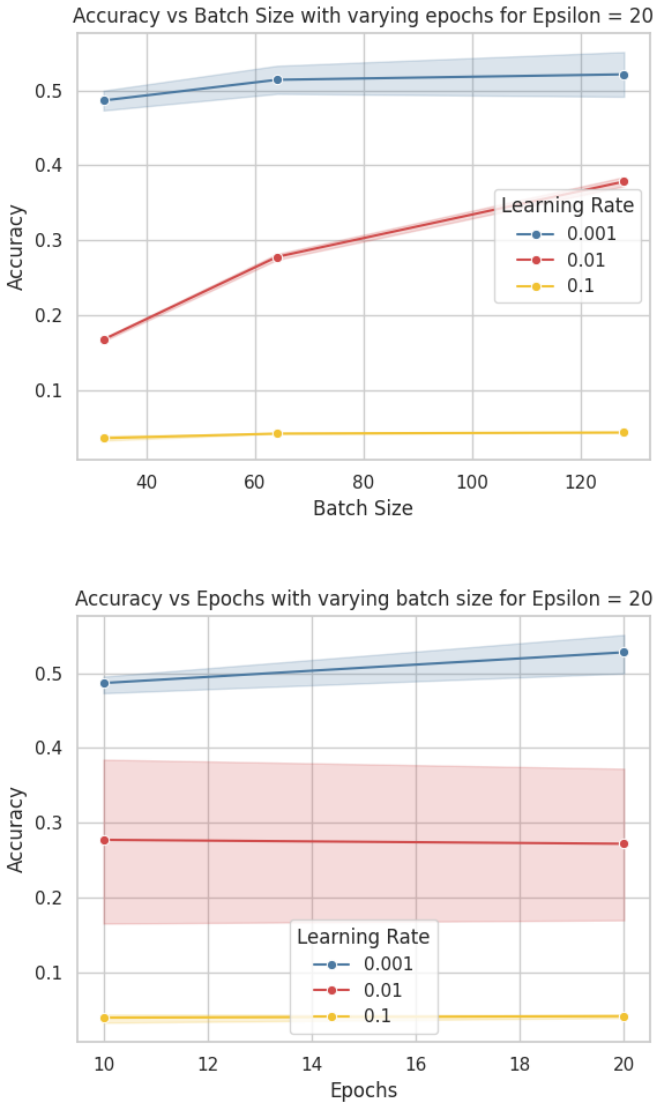


Fig. 17: Accuracy Plots to find Hyper-parameters for DPSGD and epsilon = 20

Normal Learning

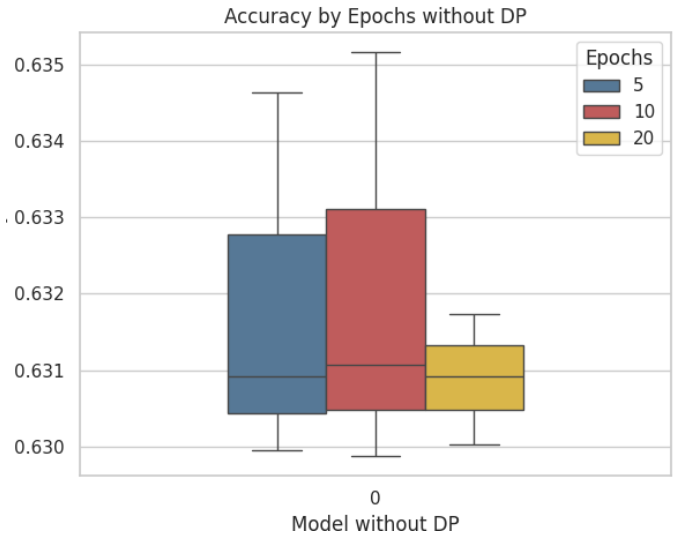


Fig. 18: Accuracy Boxplot for model without privacy

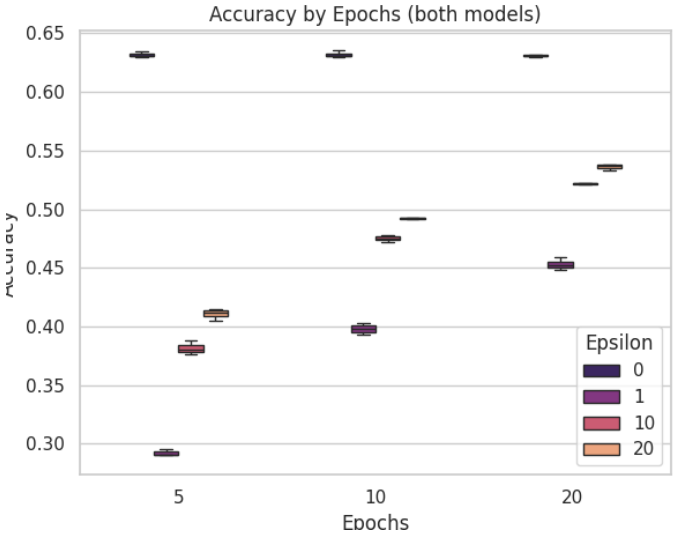


Fig. 19: Accuracy Boxplot for all models

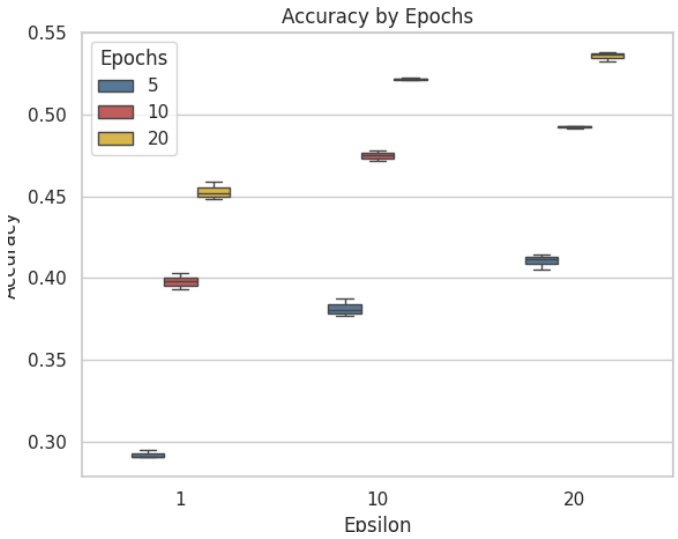


Fig. 20: Accuracy Boxplot by epochs

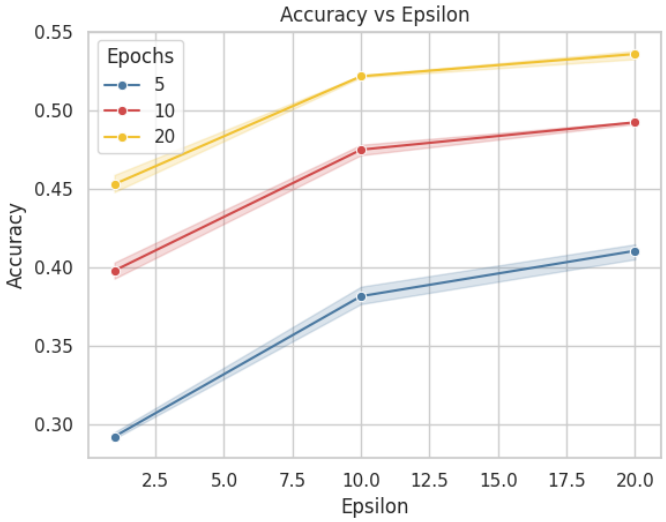


Fig. 21: Accuracy line plot by epsilon

Federated Learning

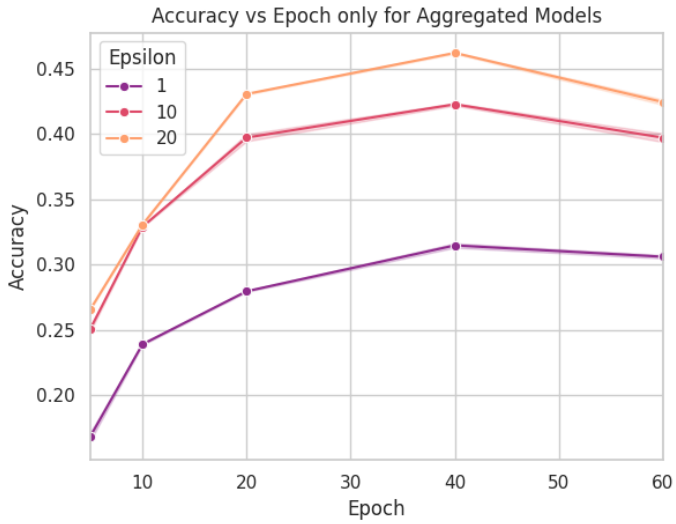


Fig. 22: Accuracy line plot by epochs