UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

# Training and Model Parameters to Defend against Tabular Leakage Attacks

Pragatheeswaran Balasubramanian

INF-3990 Master's thesis in Computer Science - May 2024

**Supervisor - Elisavet Kozyri,**
Associate Professor of Computer Science,
Department of Computer Science,
UiT The Arctic University of Norway.

**Co-supervisor - Alexander Skage,**
CEO, Finterai AS, Norway.

“Simplicity is prerequisite for reliability.”
–Edsger Dijkstra

“Beware of bugs in the above code;
I have only proved it correct, not tried it.”
–Donald Knuth

# Abstract

Federated Learning (FL) is a privacy-preserving approach to train machine learning models on distributed datasets across different organizations. This is particularly beneficial for domains like healthcare and finance, where user data is often sensitive and tabular (e.g., hospital records and financial transactions). However, recent research like Tableak highlighted vulnerabilities that can exploit information leakage in model updates to reconstruct sensitive user data from tabular FL systems.

This thesis addresses these vulnerabilities by investigating the potential of training and machine learning parameters as defensive measures against leakage attacks on tabular data.

We conducted experiments to analyze how modifying these parameters within the Federated Learning training process impacts the attacker's ability to reconstruct data.

Our findings demonstrate that specific parameter configurations, including data encoding techniques, batch updates, epoch adjustments, and the use of sequential Peer-to-Peer (P2P) architectures, can significantly hinder reconstruction attacks on tabular data. These results contribute significantly to the development of more robust and privacy-preserving FL systems, especially for applications relying on sensitive tabular data.

# Acknowledgements

I want to thank my supervisor, Elisavet Kozyri, and co-supervisor, Alexander Skage, for their constant guidance and support throughout this process.

Special thanks to Elisavet for her careful follow-up, help in formulating important questions, and collaborative brainstorming on interpreting the experiment results.

I also appreciate Alexander Skage for his invaluable guidance, providing various perspectives on approaching the problem, and generously sharing his extensive knowledge of Machine Learning and Leakage attacks with me.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

**AI** Artificial Intelligence

**FedAvg** Federated Averaging

**FedSGD** Federated Stochastic Gradient Descent

**FL** Federated Learning

**GDPR** General Data Protection Regulation

**HIPAA** Health Insurance Portability and Accountability Act

**ML** Machine Learning

**P2P** Peer-to-peer

**P2P FL** Peer-to-peer Federated Learning

**SGD** Stochastic Gradient Descent

# 1 Introduction

Imagine using your smartphone to train a medical diagnosis model using machine learning, all without ever uploading your personal health data. Federated Learning (FL) [1] is a way to train Artificial Intelligence (AI) models without sending all the data to one place. Unlike traditional centralized training, where data is aggregated into a single location for training, Federated Learning allows model training to occur directly on the devices that generate the data. Federated Learning keeps everyone's data private while allowing them to collaborate on a powerful AI model. Federated Learning has several applications in different domains like healthcare, finance, and the Internet of Things (IoT), where data privacy and security are huge concerns.

In Federated Learning, each device keeps its data private, and only updates of the model's weights are shared with a central server for training. Model weights represent the parameters of the machine learning model, such as the coefficients in a neural network, that are adjusted during training to improve the model's performance. However, this sharing process introduces vulnerability to *leakage attacks*. These attacks exploit shared updates to reconstruct user's private training data, thus compromising privacy. Recent research highlights vulnerabilities concerning tabular data formats like spreadsheets, raising concerns about privacy risks.

Previous research has investigated the privacy risks associated with exchanged updates in FL, particularly in the context of images [2], [3], [4] and text [5], [6], [7]. While these studies have offered valuable insights, they often lacked

a focus on sensitive data types. Notably, many FL applications involve tabular datasets containing personal information, such as financial details and health status. Recognizing this gap in the existing literature, the authors of the present paper [8] aimed to conduct a leakage attack specifically on tabular data. The researchers exposed the heightened vulnerability of tabular Federated Learning, revealing weaknesses in several previously considered secure setups.

Our research is built upon the findings of a foundational paper [8], which explored numerous parameters affecting leakage attacks in Federated Learning (FL) models. These parameters include batch size, Gaussian Differential Privacy (DP), epochs, network size variations, and data type. Leakage attacks exploit information disclosed during model training to reconstruct a user's private data, posing a significant threat to privacy.

This thesis explores into these vulnerabilities, focusing on how various training factors influence the reconstruction of tabular data on leakage attacks. This knowledge will be instrumental in developing more robust privacy-preserving federated learning frameworks, particularly those focused on safeguarding sensitive tabular data.

## 1.1 Research questions

Our research is founded on the following research questions:

1. How does the choice of encoding impact the accuracy of the reconstructed data, and does it also influence the overall performance of the model?

2. When training on clients, the process can occur either in a single batch or multiple batches. How does the reconstruction accuracy change with an increasing amount of batch updates during training?

3. An epoch signifies one complete pass of the training data through the machine learning algorithm. How does varying the number of batches and epochs during training impact the reconstruction accuracy?

4. In sequential Peer-to-peer (P2P) networks, how is the accuracy of reconstructed data affected if the model is trained through multiple nodes?

5. Does the model trained on reconstructed data perform as well as the model trained on original data?

These questions serve as the cornerstone of our study, directing our investigation

into the factors influencing the susceptibility of tabular data during training within the context of FL. This research aims to comprehensively understand the vulnerabilities in federated learning for tabular data by investigating how different training configurations and assumptions influence leakage attack reconstruction accuracy. The findings will be crucial in developing more robust privacy-preserving federated learning frameworks.

## 1.2 Motivation

Our motivation is driven by the practical need to better understand the trade-offs between privacy and performance in FL systems. While previous research has made significant strides in investigating parameters influencing leakage attacks, there is still a need to expand the vulnerability landscape and refine defense mechanisms to enhance privacy. Our primary aim is to contribute to the development of more resilient and privacy-preserving FL frameworks, thereby addressing a critical need in the field.

To achieve this goal, we aim to broaden the parameter space and evaluate additional factors, such as data encoding, batches, convergence, etc., that impact the effectiveness of leakage attacks. By doing so, we intend to equip practitioners and researchers with a comprehensive toolkit for optimizing FL systems and enhancing privacy safeguards.

## 1.3 Thesis outline

The rest of the thesis is organized as follows:

**Chapter 1: Introduction** Briefly introduce Federated Learning, privacy concerns, and research goals.

**Chapter 2: Background** Explain key concepts: Federated Learning, Privacy threats, Leakage attacks, and existing defenses.

**Chapter 3: Methodology** Describe the research design, experimental setup (datasets, hardware, FL architecture), and evaluation methods for effectiveness.

**Chapter 4: Experiments** Dedicate individual sections to each use case, focusing on:

1. A concise introduction and background to the specific use case.

2. Implementation details outlining any extension made to the Tableak framework to accommodate this use case.

3. Clear presentation of the experimental results using figures, tables, visualizations, and discussions.

4. Each experiment has a footnote section with GitHub commit links for the implementation code.

**Chapter 5: Conclusion** Summarize the research problem, methodology, key findings, and the overall impact of the work, along with some future works.

# 2 Background

This chapter provides essential background concepts necessary for understanding Federated Learning. We briefly overview Federated Learning, different types of algorithms, various attacks, and commonly employed defense mechanisms.

## 2.1 Federated learning

The need for Federated Learning (FL) arises from a fundamental tension between data utility and privacy. In critical domains like healthcare, finance, and legal proceedings, organizations possess vast amounts of valuable data that could be used to train robust machine learning models. However, sharing this data to a central server can compromise individual privacy, potentially leading to data breaches and unauthorized access. FL provides an elegant solution by enabling the collaborative training of a global model while keeping raw data localized on the clients' devices, be it mobile devices, edge servers, or other endpoints.

First introduced by Google in 2016, Federated Learning (FL) [1] is a decentralized machine learning paradigm that operates through a series of steps to achieve training a global model. Figure 2.1 shows the overview of Federated learning. It goes as follows:

1. The central server selects a subset of clients to participate in each round, considering efficiency. These clients can vary from individual mobile devices to organizational data centers.

2. A central server distributes a global model (initial weights) to participating devices. This model serves as a starting point for local training on each device.

3. Each device trains the global model replica on its local data. This training happens directly on the device without sending the raw data to the central server.

4. After local training, participating devices update the model weights based on their local training process.

5. The central server collects the local model updates from all participating devices in the round. Clients send model updates, typically in the form of *gradients* or *model weights*

6. The central server aggregates the received updates and applies them to the model, all without access to individual client data.

This iterative process continues, gradually refining the global model. Once training is complete, the global model can be deployed for various applications, all while preserving data privacy by ensuring that raw data remains securely localized on client devices.

### 2.1.1 Batches and epochs

In Machine Learning (ML), when dealing with large datasets, we may process the dataset by *batches* and *epochs*. These parameters could also be applied during the Federated Learning training phase. Each client can choose how to process the data and send the weights to the central server.

A batch is a small chunk of data selected from the entire dataset. During training, the model is presented with one batch at a time, processes it, and updates its internal parameters based on the information learned from that specific batch. This process is repeated iteratively, going through all the batches in the dataset until the entire dataset has been processed.

An epoch refers to a single pass or iteration through the entire dataset during the training phase. The process is repeated for a predefined number of epochs until the model converges or until a specific stopping criterion is met. With one

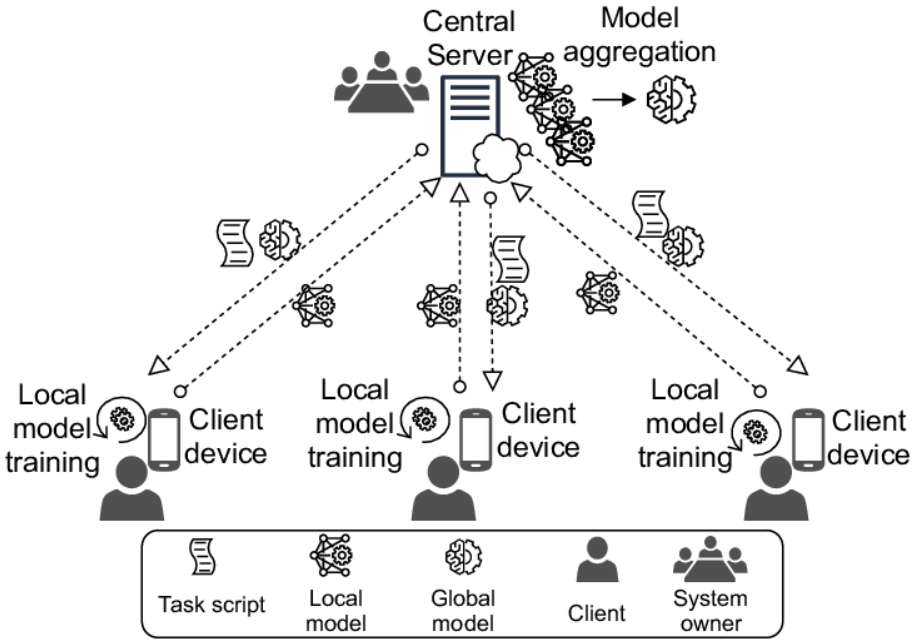


**Figure 2.1:** Federated Learning overview [9]

pass of the dataset, the model might not have enough time to learn the patterns in the data, leading to underfitting. To address this, we train the model for multiple epochs (passes).

## 2.2 Peer-to-peer federated learning

One of the recognized limitations of Federated Learning is its reliance on a central server, necessitating all participating clients to trust and communicate with a central authority. Any failure or compromise of this central server could disrupt the training process for all clients and make them vulnerable to attacks. To address this limitation and enhance privacy, a new architecture called Peer-to-peer Federated Learning was introduced.

Peer-to-peer Federated Learning (P2P FL) is a variant of traditional Federated Learning that eliminates the need for a central server. In P2P FL shown in 2.2, participating clients (peers), which can be various devices or entities, collaborate directly with one another, eliminating the need for a central server or aggregator to train a global model. Each peer in the network not only updates its local model but also shares model updates or information with other peers. While P2P FL offers more robust privacy and decentralization, it introduces challenges in coordinating communication between devices.

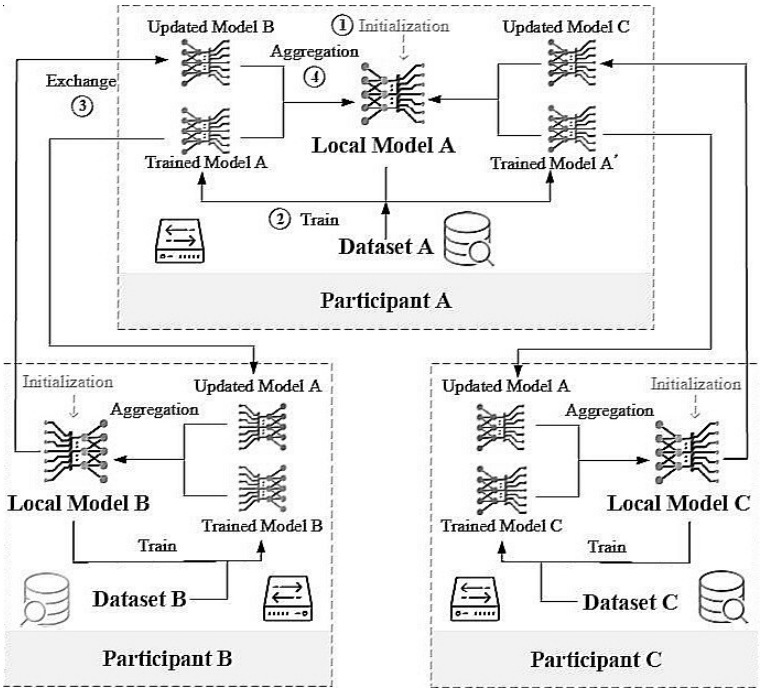


**Figure 2.2:** Federated learning architecture: Peer-to-Peer Model [10]

## Literature

Wink et al. [11] introduces an innovative approach to collaborative neural network model training in decentralized, federated environments. In this iterative process, a collective of autonomous peers conducts multiple training rounds to build a shared model jointly. Participants engage in all model training steps locally, including tasks such as stochastic gradient descent optimization, utilizing their individual private datasets. What distinguishes this approach apart is its ability to collaboratively determine a shared model without explicitly sharing the precise model weights. This is achieved through the introduction of an n-out-of-n secret sharing scheme and a specialized algorithm for calculating average values in a peer-to-peer manner. A critical facet of this approach is that it increases the communication load when the nodes try to communicate the weights.

Karras et al. [12] introduce a technique that places much importance on preserving user privacy and maintaining data integrity within distributed machine learning ecosystems. The study involves the participation of 5, 10, and 20 Raspberry Pi devices, each acting as an independent client. This approach trains each client for multiple local epochs, with subsequent aggregation of model weights. The research explores scenarios involving imbalanced and noisy data. The aim is to assess the scalability and robustness of FL in the presence of

real-world data anomalies. The study goes a step further by proposing two innovative algorithms that frame the FL scenario within a peer-to-peer context. The work introduces a client-balancing Dirichlet sampling algorithm to prevent oversampling on any individual device.

BrainTorrent [13] is a system to train a global model in a peer-to-peer manner. To start a training round, participant P first asks all other peers whether they had recently updated their local models. Peers with newer model versions send their weights to P with the respective training sample size. P then updates its local model weights by computing a weighted average of its own (old) model weights and the received (newer) model weights. The system does not rely on a central service or data pool. However, a semi-honest participant might successfully obtain training data of (all) other peers by reverse engineering the received weights. A participant could also act maliciously in many ways without being detected. For example, she or he might refuse to give honest answers regarding its current model version number and thus disproportionately profit from other peers' training efforts.

## 2.3 FedSGD and FedAvg

FedSGD and FedAvg are both training algorithms used in Federated Learning (FL). Here is a quick breakdown of each:

### 2.3.1 FedSGD (Federated stochastic gradient descent)

Federated Stochastic Gradient Descent (FEDSGD) [1] is a core optimization algorithm in Federated Learning (FL). It is a decentralized variation of the well-known Stochastic Gradient Descent (SGD) algorithm, which is widely used in centralized machine learning. The key distinction in FedSGD is that model updates, rather than raw data, are shared between the clients and the central server. The FedSGD process works in the following way: Each client conducts local model training on its own dataset, computing gradients of the loss function to the model's parameters. These gradients are then aggregated on the central server, where a global model update is computed. However, the server remains unaware of the client's data, as only the gradients are shared. This global model update is then sent back to the clients, who apply it to their local models. This cycle repeats iteratively, gradually improving the global model.

### 2.3.2 FedAvg (Federated averaging)

Federated Averaging (FedAvg) [1] is a specific algorithm for aggregating model updates in Federated Learning. In the FedAvg process, clients perform local model training and send their model weight updates instead of gradient updates to the central server. However, instead of directly averaging these updates, FedAvg employs a weighted averaging scheme, thus reducing communication and increasing convergence speed. Clients with more data or better computation capabilities are given more significant influence in shaping the global model, ensuring that less capable clients do not overshadow their insights. FedAvg has proven to be a robust and scalable aggregation algorithm, making it a vital component in the success of Federated Learning in real-world applications across a wide range of domains.

The key difference between FedSGD and FedAvg lies in their optimization approaches. FedSGD is a variant of stochastic gradient descent tailored for federated learning, where each client updates the global model using local gradients. At the same time, FedAvg aggregates model updates from decentralized clients by transmitting model weights instead of gradients.

Real-world applications typically use FedSGD when communication costs are a concern, as it involves transmitting smaller gradient updates. On the other hand, FedAvg is favored when model accuracy is a priority and communication overhead is manageable, as it can offer better performance by averaging model weights across clients.

## 2.4 Federated learning and privacy

In today's data-driven world, where machine learning advancements are transforming industries, safeguarding user privacy and data security is top priority. Privacy regulations like the General Data Protection Regulation (GDPR) in Europe and the Health Health Insurance Portability and Accountability Act (HIPAA) in the United States set strict standards for protecting sensitive personal information. Traditional machine-learning approaches often necessitate centralizing vast amounts of user data on a single server, raising concerns about data breaches and potential misuse.

Federated Learning (FL) offers a solution that enables machine learning advancements while upholding strict privacy standards. By design, FL keeps data decentralized on individual devices, be it smartphones, laptops, or other connected devices. This eliminates the need for a central data repository, significantly reducing the risk of data breaches and unauthorized access. Instead

of raw data, only model updates, essentially summaries of the local training process, are shared with a central server for aggregation. This approach ensures that sensitive information remains localized on user devices and is not transmitted over networks, minimizing exposure to potential privacy threats.

From a regulatory perspective, Federated Learning (FL) aligns seamlessly with privacy regulations like GDPR and HIPAA, which prioritize user control over personal data. By keeping data on devices, FL reduces the risk of data breaches for organizations, making it a practical and effective solution.

## 2.5 Different attacks in federated learning

Federated Learning (FL), while offering benefits in data privacy, is susceptible to various attacks that can compromise the integrity and confidentiality of the training process. Here is an overview of some common FL attacks:

1. **Data Poisoning Attacks:** These attacks involve injecting false or misleading information into the training data to distort the model's learning. Attackers often achieve this by strategically introducing biased or malicious samples, aiming to influence the model updates in their favor and ultimately degrade the model's performance [14].

2. **Model Poisoning Attacks:** These attacks target the central model itself, aiming to disrupt the entire FL process by manipulating the aggregated updates. Attackers typically select poisoned updates that, when combined, degrade the global model's performance for all participants [15].

3. **Inference Attacks:** These attacks exploit information leaks within model updates to deduce sensitive details about individual users, such as their location or health conditions [16]. By analyzing the updates, attackers can infer information about specific users or their data, compromising privacy.

4. **Backdoor Attacks:** These attacks embed hidden functionality within the trained model, allowing attackers to manipulate its output for specific inputs while maintaining normal functionality for most tasks [17]. Attackers often insert backdoors during training, enabling them to trigger specific behaviors in the model with carefully crafted inputs.

5. **Evasion Attacks:** These attacks manipulate data sent to the model to avoid detection, allowing attackers to benefit from the model without being noticed. This can involve crafting adversarial examples that exploit

vulnerabilities in the model's decision-making process, enabling attackers to, for example, obtain loan approvals without meeting the valid requirements [18].

6. **Model Inversion Attacks:** These attacks attempt to reconstruct the original user data from the trained model, posing a significant privacy breach [19]. Attackers leverage model inversion by querying the model with inputs designed to reveal sensitive information, thereby potentially reconstructing aspects of the original training data.

We will focus on leakage attacks, which are part of model inversion attacks. *Leakage attacks* focus on extracting information from the exchanged updates in FL training. While the specific methods might differ, the ultimate goal aligns with model inversion attacks: to gain access to sensitive information that was originally used to train the model.

## 2.6 Leakage attack and reconstruction

### 2.6.1 Gradient inversion attack

Most data-recovery attacks, called gradient inversion attacks, targets FedSGD where devices only send a single update (gradient) to a central server. For a long time, many researchers believed sharing gradients was safe because they did not reveal the original training data. However, the research "Deep Leakage from Gradients (DLG)" [2] showed that attackers could potentially steal private training data just from these gradients. Here is how attackers can potentially steal private training data by analyzing the shared gradients.

1. We start by creating fake data (called "dummy inputs and labels"). Then, we perform the training steps a model normally does on this fake data, generating "dummy gradients."

2. Instead of using these dummy gradients to train the model as usual, we do the opposite. We adjust the fake data to make its gradients match the real gradients from the actual training data.

3. By repeatedly refining the fake data to match the real gradients better, we can eventually uncover the original training data used to create those gradients. This includes both the original inputs (data points) and the labels (categories) associated with them.

Note that this fake dataset is like a puzzle being solved. The objective is to

adjust this fake data until it produces updates that closely resemble the real updates sent by the user's device.

### 2.6.2 Gradient inversion in federated averaging

While gradient leakage attacks can successfully retrieve data even in scenarios involving complex datasets, their practical utility is constrained. In real-world applications, the model is often trained using FedAvg [1] algorithm. This study [20] represents one of the initial research dedicated to leakage attack systems that use the FedAvg Algorithm. Please note this attack also falls under Gradient Inversion category but use weights instead of gradients. Here is how the attack works:

1. **Estimating label counts:** The attacker first determines how many times each category (label) appears in a user's data by analyzing the updates. This is done by looking for patterns in the updates and making guesses (interpolating).

2. **Reconstructing individual data points:** Once a rough idea of the label count is obtained, the attacker uses a special "automatic differentiation" technique to simulate the training process on a fake dataset. This fake dataset is adjusted until it produces updates that closely resemble the real updates. To achieve this, consideration is given to two things:

   (a) **Changing model parameters:** Accounting for the fact that the model being trained keeps getting updated during training, adjustments need to be made to the fake data accordingly.

   (b) **Large number of batches:** Since training occurs in "batches" (smaller chunks of data), a special method (epoch order-invariant prior) is used to account for the fact that there might be many batches of data used for training.

3. **Recovering labels:** In some cases, the attacker might also attempt to recover the exact labels associated with each data point. This is done using techniques similar to those used in other attacks on the FedSGD algorithm.

Gradient leakage and Federated averaging attacks are operated under *Honest-But-Curious* server model. The server (attacker) faithfully follows the established communication protocol for FL, meaning it does not change the data or deviate from the agreed-upon training process. However, the server attempts to extract as much information as possible from the received updates.

## 2.7 Domain specific leakage attack

The existing literature highlights that, in image and text domains, specific characteristics of the data can be exploited to achieve high-accuracy reconstruction.

In image processing and computer vision, an *image prior* represents assumptions or knowledge we have about what a typical image should look like. In the context of FL attacks on images, attackers can leverage image priors to improve the accuracy of reconstructing user data from model updates [3, 4]. These priors can be:

1. **Statistical:** Based on statistical properties of natural images, such as smoothness, local coherence (neighboring pixels having similar values), and color distributions.

2. **Structural:** Knowledge about common image structures, like edges, textures, and objects.

*Pre-trained language models* (PLMs) are powerful neural network models trained on massive amounts of text data. They learn complex relationships between words and can perform various tasks like machine translation, text summarization, and question answering. In FL attacks on text data, attackers can exploit PLMs to reconstruct user-generated text from model updates [6, 7].

1. **Semantic Understanding:** PLMs can analyze the relationships between words and concepts within the updates, helping the attacker reconstruct a text that is grammatically and semantically correct.

Existing image and text solutions would not translate well to tabular data. Tabular data typically has a mix of discrete (categorical) and continuous (numerical) features, unlike images (continuous) and text (discrete). This mixed nature makes it difficult to judge the validity of a reconstructed data point. Vero et al. [8] solve this problem by running separate optimization processes with different initializations and then combining their results through feature-wise pooling [8]. This way, we use the structure of tabular data to combine independent reconstructions for a better and more reliable final estimate of the true data.

## 2.8 Defenses for leakage attacks

### 2.8.1 Gradient perturbation

Gradient perturbation [21] involves adding controlled noise to the gradients during the training process. This noise helps prevent adversaries from accurately discerning the contribution of individual clients to the global model updates. By introducing perturbations to the gradients, this mechanism aims to obscure specific details of individual client contributions, making it more challenging for adversaries to perform precise inversion attacks.

### 2.8.2 Gradient compression

Gradient compression [21] involves reducing the size of the gradients before transmission from clients to the central server. This compression is typically performed to preserve the general information while minimizing the risk of privacy leakage. Compressing the gradients addresses potential privacy risks associated with transmitting detailed gradient information, making it more difficult for adversaries to extract sensitive information. Limits the magnitude of gradients during training to improve privacy. However, it can also unintentionally introduce noise.

### 2.8.3 Secure Multi-Party Computation

Secure Multi-Party Computation (SMPC) [22] is a cryptographic technique that enables multiple parties to collectively compute a function over their inputs while keeping the privacy of those inputs. In the context of FL, it enables the aggregation of model updates without revealing the individual updates. SMPC ensures that the aggregation process is performed securely, preventing adversaries from gaining insights into the specific contributions of individual clients to the global model. However, it has some significant drawbacks, particularly regarding computational efficiency and latency.

### 2.8.4 Differential privacy

Differential privacy [23] involves injecting random noise into the computations to provide a mathematically rigorous guarantee of privacy. It aims to ensure that the inclusion or exclusion of any single client's data does not significantly impact the overall output. By applying differential privacy, FL systems can offer a strong privacy guarantee, making it challenging for adversaries to infer specific details about individual clients' contributions or data. Adding noise or

clipping gradients can obscure the underlying patterns in the data, making it harder for the model to learn effectively. This can lead to an overall reduced model accuracy.

# 3

# Methodology

This chapter outlines the methodology employed in our research for defense against leakage attacks targeting tabular data in FL. We will discuss the experimental setup and evaluation of the results.

## 3.1 Foundation

Our work expands upon the existing research conducted by [8] with the open-source TabLeak code. Using it as a foundational framework, we significantly enhance its capabilities to accommodate our expanded set of experiments.

The research adopts an experimental approach. We will conduct a series of experiments to systematically evaluate the impact of new factors related to deep learning on reconstruction accuracy. Our primary objective is to address the research questions that significantly influence the ability to reconstruct private user data from shared model updates. This comprehensive exploration aims to identify critical factors affecting the reconstruction process's accuracy in the Federated Learning setup.

## 3.2 Experimental setup

The following conditions apply to all experiments unless explicitly stated otherwise:

**Number of nodes:** Two nodes are used in the experiments:

- **Victim**: This node trains a machine-learning model for federated learning and unknowingly leaks information through gradient updates.
- **Attacker**: This node aims to gain information about the victim's private dataset.

**Attack method:** The attacker leverages the gradients sent by the victim during the training process. Gradients provide information about how the victim's model is changing in response to the training data.

**Attacker model:** The attacker operates under the "honest-but-curious" assumption. This means the attacker faithfully follows the communication protocol but tries to learn as much as possible about the victim's model using the available information (gradients).

**Neural network training:** The entire dataset is not trained in federated learning. Instead, the victim trains on one batch of data. After each training step, the victim sends the gradients associated with that batch to the attacker.

In the attacker node, we use the Adam optimizer with a learning rate of 0.06 for 1500 attack iterations and without a learning rate schedule to perform the optimization. We attack a fully connected non-linear neural network with ReLU activation with two hidden layers of 100 neurons each.

**Datasets:** All experiments were carried out on two popular mixed-type tabular binary classification datasets, the Adult census dataset [24] and the Law school Admission dataset [25].

## 3.3 Validation and evaluation metrics

### 3.3.1 Reconstruction accuracy and 0-1 Loss

In the context of Federated Learning (FL) leakage attacks targeting tabular data, *reconstruction accuracy* refers to the success rate of an attacker in retrieving or inferring the original training data points from the model updates exchanged

during the FL process. Reconstruction accuracy is typically measured using a metric like 0-1 loss [8]. This metric compares the reconstructed data points with the original data points and assigns a score based on how many features (attributes) match perfectly. A higher reconstruction accuracy indicates that the attacker was more successful in retrieving the original data points from the model updates.

Here is how $0-1$ loss works. We compare each value between the corresponding positions in the ground truth and reconstructed data. If the element in the original data perfectly matches the element in the reconstructed data, the loss for that element is 0. If the element in the original data does not match the element in the reconstructed data, the loss for that element is 1. Here is the formula:

$$0-1_{\text{loss}} = \frac{\text{Number of matching elements}}{\text{Total number of elements}}$$

Here is how to interpret it:

1. A higher $0-1$ loss value (closer to 1) indicates a lower reconstruction accuracy. This means the attacker's reconstructed data has many mismatches compared to the original data.

2. Conversely, a lower $0-1$ loss value (closer to 0) indicates a higher reconstruction accuracy. This suggests that the attacker was more successful in retrieving the original data with more matching elements.

A quick example: Imagine the table 3.1 where "Ground Truth" is the original data the global model is trained with, and "Reconstructed Data" is data constructed by the leakage attack. Comparing them, we can see $0-1$ loss,

| Ground Truth | Reconstructed Data | 0-1 Loss |
|---|---|---|
| Product | Product | 0 |
| Shirt | Shirt | 0 |
| Book | Hat | 1 |
| Headphones | Headphones | 0 |
| Movie | Book | 1 |

**Table 3.1:** $0-1$ Loss

Hence, the overall 0-1 Loss for this example is:

$$\frac{\text{Number of matching elements}}{\text{Total number of elements}} = \frac{2}{5} = 0.4$$

A $0-1$ loss of 0.4 implies that the attacker's reconstruction achieves a 60% reconstruction accuracy based on perfectly matching elements.

### 3.3.2 Model accuracy

In machine learning models, *Model accuracy* is used as a performance metric representing the proportion of correctly classified instances in a dataset. It is calculated using the formula:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

It is a crucial metric to evaluate how well a trained neural network performs on unseen data. A higher model accuracy indicates that the neural network is making a higher proportion of correct predictions on the unseen data. This suggests the model has learned the training data patterns and can generalize well to new examples. While a high model accuracy is desirable, it is not the only factor to consider. Other metrics like precision, recall or F1-score might be more relevant depending on the application. Model accuracy can be misleading if the dataset is imbalanced (unequal distribution of classes)—more on this in upcoming sections.

In the ideal Federated Learning (FL) scenario, we want two things:

1. **Higher Model Accuracy:** A model that learns well from all the data without seeing it directly.

2. **Lower Reconstruction Accuracy:** To keep user data private from potential attacks.

Unfortunately, these objectives can sometimes conflict. Approaches that enhance the model's accuracy by sharing more information through updates can also make it simpler for attackers to decipher users' data, resulting in higher reconstruction accuracy. Striking a balance between Reconstruction accuracy and Model accuracy is a pivotal objective of our research. We aim to pinpoint the optimal trade-off point between these two metrics. This equilibrium is crucial, as it empowers us to make informed decisions about parameter choices, model training strategies, and the overall design of FL systems.

While accuracy is a straightforward and easy-to-understand metric, it can be misleading. It provides a clear indication of how well the model performs on the specific dataset, but it's not foolproof. If the dataset is unbalanced (i.e.) heavily skewed towards a particular class, the model might achieve high accuracy by

simply predicting that class for everything. This doesn't necessarily imply that the model generalizes well to unseen data. Depending on the application, other factors, like the cost of mistakes, might be crucial. In medical diagnosis, false negatives (missed diseases) are often graver than false positives (incorrect diagnoses of illness). It's important to consider these nuances when evaluating model performance.

### 3.3.3 Model performance

Model performance is a concept that encompasses various metrics to evaluate a model's effectiveness. To calculate model performance, we need to understand *confusion matrix*. Table 3.2 creates a 2x2 grid that visually represents the performance of a classification model on a test dataset. It has two rows (representing actual classes) and two columns (representing predicted classes). The confusion matrix helps identify which classes the model is most likely to misclassify for one another. Here is a breakdown of the terms used in the formulas and the confusion matrix:

| **Actual** | **Predicted** | |
|---|---|---|
| | **Positive** | **Negative** |
| **Positive** | TP | FN |
| **Negative** | FP | TN |

**Table 3.2:** Confusion Matrix

1. **TP (True Positive):** This value indicates the number of instances where the model correctly identified the positive class.

2. **FP (False Positive):** This value indicates the number of instances where the model incorrectly identified the positive class (e.g., identifying a negative instance as positive).

3. **TN (True Negative):** This value indicates the number of instances where the model correctly identified the negative class.

4. **FN (False Negative):** This value indicates the number of instances where the model incorrectly identified the negative class (e.g., missing a positive instance).

For example, it would look something like this,

$$\begin{bmatrix} 100 & 10 \\ 5 & 85 \end{bmatrix}$$

1. TP (100): The model correctly classified 100 instances that belong to the positive class.

2. FP (10): The model incorrectly classified 10 instances as positive that belong to the negative class.

3. FN (5): The model incorrectly classified 5 instances as negative that belong to the positive class (missed positives).

4. TN (85): The model correctly classified 85 instances that belong to the negative class.

Using the values from the confusion matrix, you can calculate various performance metrics:

1. **Accuracy:** This represents the overall proportion of correct predictions.

   $\frac{TP+TN}{TP+TN+FP+FN}$

2. **Precision:** This measures the proportion of correct positive predictions.

   $\frac{TP}{TP+FP}$

3. **Recall:** This measures the proportion of actual positive cases that were identified correctly by the model.

   $\frac{TP}{TP+FN}$

4. **Specificity (TNR):** This measures the proportion of actual negative cases the model correctly identified.

   $\frac{TN}{TN+FP}$

5. **F1-Score:** F1 score combines precision and recall into a single metric, providing a balanced view of model performance.

   $2 \times \frac{Precision \times Recall}{Precision+Recall}$

Following our confusion matrix example, we can analyze our model performance:

1. **High Accuracy:** The model seems to perform well regarding overall accuracy based on confusion matrix. With 185 correctly classified instances (100 TP + 85 TN) out of a potential 200, the model achieves an accuracy

of 92.5%.

2. **Good at Identifying Positives:** The high number of True Positives (100) indicates that the model correctly identifies instances that belong to the positive class.

3. **Low False Negatives:** The relatively low number of False Negatives (5) suggests the model does not miss many positive cases.

4. **Some Confusion with Negatives:** While the accuracy is high, the 10 False Positives indicate the model is occasionally mistaking negative instances for positive ones.

We will primarily use model accuracy as our evaluation metric to prioritize efficiency. However, a select few experiments will involve a more comprehensive performance analysis.

# 4

# Experiments

This chapter will address the problem statement by conducting experiments on each research question and discussing the results.

## 4.1 Encoding of dataset

**Question:** How does the choice of encoding impact the reconstruction accuracy of the attack?

### 4.1.1 Background

Encoding is a crucial preprocessing step in machine learning datasets, mainly when dealing with categorical variables. A categorical variable is a variable that has a finite number of distinct values. Examples of such variables are sex, race, or education level. Most machine learning algorithms, including deep learning neural networks, require input and output variables to be numeric. Categorical variables cannot be used directly in machine learning algorithms and need to be converted into a numerical format for machine learning algorithms to learn from the data effectively. Several encoding techniques can be used to convert categorical variables into numerical variables.

The most commonly used encoding type in machine learning is *one-hot encoding*.

In one-hot encoding, each categorical value is represented by a vector with a single "hot" (1) position, resulting in a binary vector where only one element is active at a time. For example, in the "Color" variable (Table 4.1), there are 3 categorical values, and therefore, we have 3 possible binary values. This encoding assigns a "1" value to the bit corresponding to the color, while "0" values are assigned to the remaining bits. For instance, Red would have a "1" in the first position, Green would have a "1" in the second position, and so forth.

| Color | b1 | b2 | b3 |
|---|---|---|---|
| **Red** | 1 | 0 | 0 |
| **Green** | 0 | 1 | 0 |
| **Blue** | 0 | 0 | 1 |

**Table 4.1:** One-hot encoding

*Binary encoding* is also used for encoding categorical variables in machine learning. In this encoding, categorical values are converted as integers and then converted into equivalent binary code. Then, the digits from the binary code are used as input for the neural network. For example, in the "Color" variable (Table 4.2), there are 3 categorical values, and therefore, we have 3 possible binary values. First, we convert each value to an integer like 0, 1, and 2 for the respective colors. Then, we replace these integers with equivalent binary code. For example, for the value "Color"=Red, we can assign an integer of 0, which can then be converted to a binary code of 00.

| Color | Value | b1 | b2 |
|---|---|---|---|
| **Red** | 0 | 0 | 0 |
| **Green** | 1 | 0 | 1 |
| **Blue** | 2 | 1 | 0 |

**Table 4.2:** Binary encoding

One significant advantage of employing one-hot encoding instead of binary encoding is its ability to prevent ambiguity in machine learning algorithms. By converting categorical values into numerical representations, such as red=0, green=1, and blue=2, we risk introducing potential confusion, as the algorithm may interpret numerical values as ordinal. Ordinal data have a natural order or rank based on some hierarchical scale, like high to low. However, with one-hot encoding, each category is represented by a binary vector, ensuring no ordinal relationship is assumed and avoiding this confusion.

The key difference between one-hot encoding and binary encoding in machine learning is their representation of categorical variables. One-hot encoding represents each category using a binary vector, with only one element being

"hot" (1), while the others are "cold" (0). Binary encoding uses binary digits to represent each category, potentially resulting in fewer dimensions. In the above example, one-hot encoding needed 3 bits, while binary encoding needed only 2 bits.

When dealing with categorical variables, which are binary encoded, not all possible binary values may be utilized, leading to potential inefficiencies. For instance, in the above example (Table 4.2), if there are only three categories (00=Red, 01=Green, 10=Blue) and binary encoding is used with two bits, the binary representation "11" is not utilized. With a binary encoded dataset, if an attack optimization algorithm generates a binary string out of the valid range, we must handle it by implementing a clipping mechanism.

The clipping mechanism aims to bring any out-of-range binary string back into the valid set of binary strings within range, which can then be mapped to a categorical value. This clipping process involves converting the binary string to an integer and then scaling the number to fit within the acceptable range. The scaling is performed based on the count of possible values, deviating from the conventional $2^n$ representation, ensuring the integrity and consistency of the data. In general, to scale the variable $x$ into a range $[a, b]$, you can use min-max normalization [26]:

$$x_{\text{scaled}} = a + \frac{(x - \text{old\_min}) \cdot (b - a)}{\text{old\_max} - \text{old\_min}}$$

This formula is used to map the original variable $x$ from the old range $[\text{old\_min}, \text{old\_max}]$ to the new range $[a, b]$.

Suppose the attack optimization algorithm generates 11 as a binary code that falls out of the desired range. In that case, the clipping mechanism adjusts this number to an integer value within the acceptable range. When converting the binary representation 11 to the integer 4, representing $x$ in the old range $[0, 4]$ to the new range $[0, 3]$, the scaling formula is applied as follows:

$$x_{\text{scaled}} = 0 + \frac{(4 - 0) \cdot (3 - 0)}{4 - 0} = \frac{4 \cdot 3}{4} = 3$$

So, the scaled value of $x$ is 3, which is then converted to 10 binary code, which can be mapped to a valid categorical value.

### 4.1.2 Implementation

Within the current codebase, *BaseDataset* is a class used to parse raw data (CSV, data file, etc.) into input dataset objects for the machine learning algorithm. The type of encoding of categorical values is implemented here. A new base

dataset tailored for binary encoding is introduced, leveraging the functions *to_numeric_binary* and *to_categorical_binary* to encode and decode data into binary format. Datasets can inherit the *BinaryBaseDataset* to make the encoding scheme of categorical values as binary strings instead of a one-hot string.

During binary encoding, all possible categorical variables are mapped to binary strings based on their order (For example, table 4.3). Then, on the dataset, each categorical value is substituted with its corresponding binary string from the map during encoding to be used for neural network input. Upon decoding, typically, to compare the attack result with ground truth, a value that falls outside the expected range is adjusted using the formula mentioned earlier (For example, table 4.4). Finally, the established map converts the binary strings into actual categories.

| Color | Binary Representation |
|---|---|
| Red | 00 |
| Green | 01 |
| Blue | 10 |

**Table 4.3:** Color-to-Binary Mapping

| Binary Code | Decoded Color |
|---|---|
| 00 | Red |
| 10 | Blue |
| 01 | Green |
| 11 (Invalid) | Blue (After Clipping - 10) |
| 00 | Red |

**Table 4.4:** Optimization Algorithm Output with Clipping

Now that the implementation is done, we employed four distinct datasets for the experiment: ADULT (One-hot), ADULT (Binary), LawSchool (One-hot), and LawSchool (Binary). The binary-encoded variants of the ADULT (One-hot) and LawSchool (One-hot) datasets are denoted as ADULT (Binary) and LawSchool (Binary), respectively. During the experiments, we systematically varied the batch size, ranging from 4 to 128, while conducting the training. This standardized methodology was employed to comprehensively assess the impact of binary encoding on reconstruction accuracy, mainly how it changes with increasing batch sizes. We are particularly interested in measuring reconstruction accuracy to understand how binary encoding affects the attacker's ability to retrieve private user data from the model updates.

* https://github.com/leopragi/tableak_fl/commit/encoding

### 4.1.3 Result

Figure 4.1 illustrates the reconstruction accuracy for different datasets (ADULT and Lawschool) and encoding types (One-hot and Binary) across various batch sizes. For both ADULT and LawSchool datasets, the binary encoding consistently shows lower reconstruction accuracy across all batch sizes compared to the one-hot encoded dataset.

Binary encoding reduces reconstruction accuracy because it represents categorical variables as binary vectors, where each category is encoded as a unique combination of binary digits. In contrast to one-hot encoding, where each category has its dimension, binary encoding compresses the information into fewer dimensions, potentially leading to loss of information. This compression can result in ambiguity during the reconstruction process, making it challenging to decode accurately.

Our results suggest that binary encoding might favor privacy-preserving federated learning. Lower reconstruction accuracy compared to One-hot encoding in leakage attacks indicates attackers have difficulty reconstructing user data. One-hot encoding, while convenient for training, might leak more information. However, further analysis of other factors, such as model accuracy, needs to be considered to quantify the benefits of binary encoding.

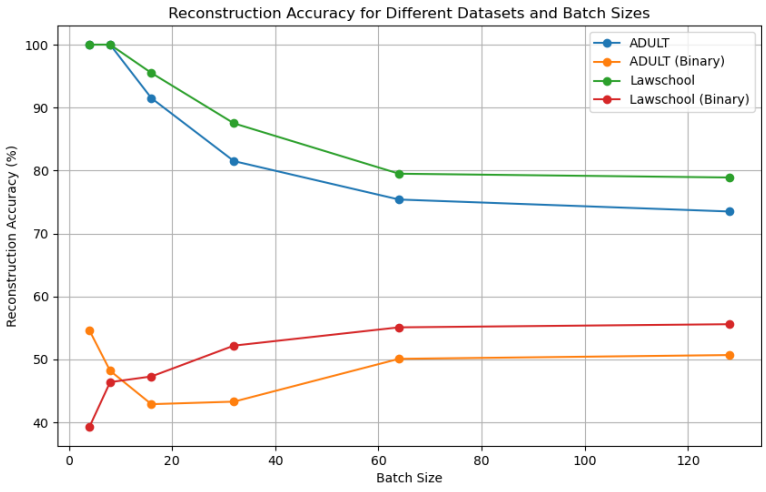


**Figure 4.1:** Reconstruction accuracy of datasets with one-hot and binary encoding

## 4.2 Model accuracy on binary encoded dataset

**Question:** How does binary encoding affect the model accuracy?

### 4.2.1 Background

In an encoded dataset, *dimensionality* refers to the number of features or dimensions after applying an encoding technique. In our previous experiments, we used binary encoding to reduce the reconstruction accuracy of leakage attacks in federated learning systems. By converting categorical variables into binary format, we reduced the vector space in data representation, potentially making it harder for attackers to reconstruct user data. However, there is a crucial trade-off to consider.

While binary encoding offers potential privacy benefits, it also impacts the model's learning process due to the reduced dimensionality because of reduced expressive power. So, it becomes crucial to evaluate the model's accuracy and dimensionality, considering this binary-encoded data representation.

### 4.2.2 Implementation

In our experiment, we employed four of the same datasets we used in the encoding experiment: ADULT (One-hot), ADULT (Binary), LawSchool (One-hot), and LawSchool (Binary). This approach involves training a neural network model and assessing its accuracy for both types of encoding of the datasets. Throughout the experiments, we maintained a fixed batch size of 500 and conducted training for a single epoch for the entire dataset. This standardized approach allowed us to systematically evaluate the performance of our model across different datasets and experimental conditions.

### 4.2.3 Result

Table 4.5 shows model accuracy and dimensions of the two datasets in different encoding and dimensions. Generally, the model accuracy remained stable across different dataset types, highlighting the robustness of the model across varied data domains. Despite the dimensionality reduction achieved through binary encoding compared to one-hot encoding, the differences in model accuracies are not substantial.

* https://github.com/leopragi/tableak_fl/commit/encoding_accuracy

This observation underscores the effectiveness of the non-linear ReLU neural network architecture in handling the reduced feature space without significant compromise in predictive performance [27]. Notably, while the reconstruction accuracy may decrease with dimensionality reduction through binary encoding, the model's overall performance remains well maintained.

| Dataset | Encoding | Dimensions | Model Accuracy |
|---|---|---|---|
| **ADULT** | Binary | 36 | 86.35% |
| | One-hot | 108 | 85.92% |
| **LawSchool** | Binary | 17 | 87.96% |
| | One-hot | 39 | 88.09% |

**Table 4.5:** Model Performance Comparison between Binary and One-hot Encoding

Our experiments suggest that binary encoding might be a promising approach for privacy-preserving federated learning. Compared to one-hot encoding schemes, binary encoding appears to achieve lower reconstruction accuracy in leakage attacks and does not affect the model accuracy. This indicates that attackers have more incredible difficulty retrieving private user data when models are trained on binary-encoded data.

### 4.2.4 Discussion and future work:

Binary encoding reduces the data dimensionality, which might affect the model's ability to learn intricate patterns in specific datasets. Further analysis of the model, Precision, Recall, and Specificity is needed. Also, exploring various encoding schemes like Label Encoding, Ordinal Encoding, Target Encoding, etc., can be beneficial, as some might offer superior privacy preservation without significant sacrifices in model performance.

## 4.3 Multiple batches on client-side

**Question:** When training with multiple clients, on each client, the process can occur either in a single batch or multiple batches. How does the reconstruction accuracy change with increasing amount of batch updates during training?

### 4.3.1 Background

The original paper focused on reconstructing the dataset using the FedAvg attack algorithm when training with multiple batches on a single node. Experiments varied in the number of local batches and epochs for a client dataset size of 32. These experiments demonstrated a significant decrease in reconstruction accuracy when using multiple batches.

However, these experiments did not consider gradient inversion (for FedSGD) attacks on batches. We aim to investigate how the number of local batches used during training affects the success rate of Gradient Inversion attacks in federated learning.

### 4.3.2 Implementation

In a centralized federated learning setup, we can follow the following process to process multiple batches on the client side. During a training round, each client iterates through its local dataset one batch at a time. For each batch:

1. The device feeds the current batch through a copy of the global model (received from the central server in the previous round).

2. It performs a forward pass, calculating the model's predictions on the batch data.

3. It then performs a backward pass, analyzing the difference between predictions and actual labels to compute gradients. These gradients are applied to the model and also stored in a list.

This process repeats until all batches are trained. Once the client finishes processing all its batches, it calculates an average of the gradients accumulated across all its local batches. Then, the client transmits the averaged gradients to the central server. The central server collects gradients from all participating clients in the round, and then it can perform an inversion attack on received gradients. Since the client shares gradients instead of weights, the attacker node can use a gradient inversion attack.

The client stores the gradient updates in our code on a list after each local batch training. Ultimately, it calculates the average of all the gradients accumulated across all its local batches. We name this function as $average\_gradients$, implemented in the $FullyConnectedTrainer$ module.

For example, let us say the client has 1000 total records for training. Let us

set the local batch size (*l_batch_size*) to 8. The client will process the data in 125 batches (1000 records / 8 batches/record = 125 batches). Gradients are calculated and accumulated in a list during training on each batch. Once all batches are processed, the *average_gradients* function is used to compute the average of all the accumulated gradients from the 125 batches. This single averaged gradient is then sent to the central server to contribute to the global model update. While doing the attack, the attacker node will try to reconstruct a total number of records, which is 1000 instead of batch size 8.

To facilitate the comparison of relative performance on increasing batches, we have designed the experiment to process 1024 records, dividing them into four different batch splits: 1 batch, 8 batches, 16 batches, and 64 batches. This will allow us to see how the reconstruction rate is affected relatively by the increasing number of batches with the same number of records. This approach allows us to systematically investigate how varying numbers of batches influence the reconstruction accuracy of leakage attacks.

### 4.3.3 Result

Figure 4.2 reveals a critical finding: as the number of batches used for training increases, the accuracy of gradient inversion attacks significantly declines. Our experiment began with a baseline scenario where all 1024 records were processed as a single batch. In this case, the attacker achieved a certain level of reconstruction accuracy. However, when we divided the same data into smaller batches (8 batches of 128 records each), the reconstruction accuracy dropped by approximately 10%. This trend continued as the number of batches further increased. This highlights a crucial observation: using multiple batches during training makes it significantly harder for attackers to reconstruct private user data from the exchanged updates than a single batch with the same number of records.

The original paper presented its findings on reconstruction accuracy based on the number of updates combined with variations in batches and epochs. For example, 4 batches with 10 epochs is 40 updates. We must factor in epochs alongside the number of batches to ensure a fair comparison.

### 4.3.4 Discussion

Experiments using multiple batches during training revealed that increasing the number of batches significantly reduces reconstruction accuracy. While

* https://github.com/leopragi/tableak_fl/commit/multi_batches

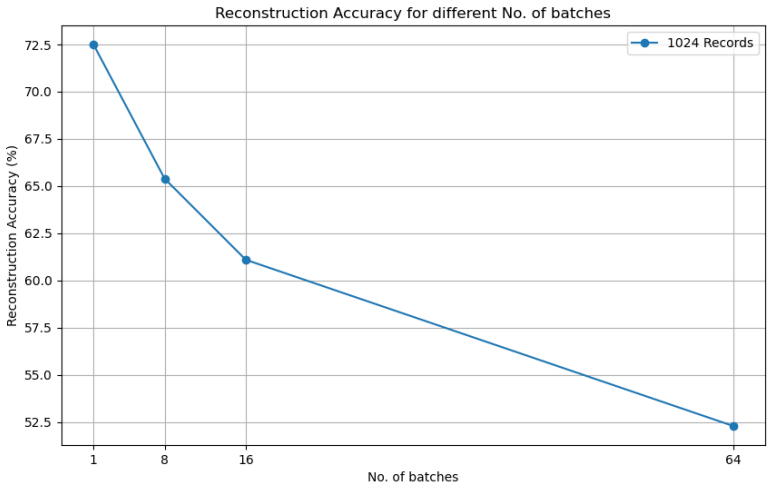


**Figure 4.2:** Reconstruction accuracy with increasing no. of batches

increasing the number of batches enhances privacy, it can also lead to increased training time and communication overhead between participating devices. There is a crucial trade-off between achieving a desired level of privacy and maintaining efficiency in the training process.

## 4.4 Multiple batches with epoch on client-side

**Question:** How does varying the number of batches and epochs during training impact the reconstruction accuracy?

### 4.4.1 Background

In Federated Learning (FL), the concept of epochs remains similar but with some distinctions due to the distributed nature of the training data. In FL, each participating device or client trains its local model on its dataset for a specified number of local epochs. After the local training, the updated model parameters are aggregated across all clients to create a global model.

In our previous experiments, we observed a decrease in reconstruction accuracy as the number of batches increased. We aim to extend our investigation by introducing the epochs and examining their impact on reconstruction accuracy in addition to the number of batches.

### 4.4.2 Implementation

We are using the *FullyConnectedTrainer* module from the existing code base to conduct the experiment. *FullyConnectedTrainer* takes a neural network and data as input and trains the network iteratively. Specifically, we are interested in local batch size (*l_batch_size*) and *epochs* on the training process, which helps us to control the number of records that need to be trained in a whole batch and epochs for the number of passes for the dataset. Specifically, we will explore the impact of 1, 5, and 10 epochs with increased batches. Setting the 1 epoch as the baseline will allow us to observe the effects of the other two configurations.

### 4.4.3 Result

Figure 4.3 shows that higher epoch configurations generally lead to lower reconstruction accuracy across all batch sizes. Additionally, larger batch sizes tend to result in lower reconstruction accuracy regardless of the epoch configuration. Larger batch sizes with higher epochs do not achieve much lower reconstruction accuracy. This trend suggests that combining larger epochs and larger batch sizes yields lower reconstruction accuracy in leakage attacks of federated learning when used moderately.

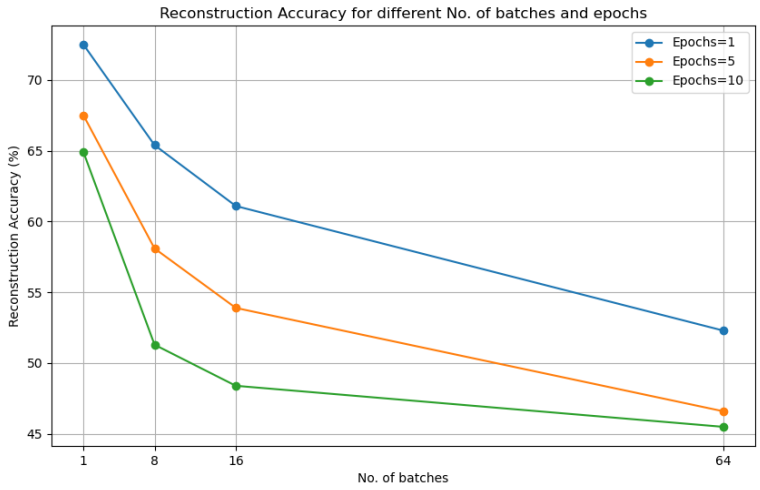


**Figure 4.3:** Reconstruction accuracy with increasing no. of batches and epochs

* https://github.com/leopragi/tableak_fl/commit/multi_batches_epochs

## 4.5 Sequentially connected peer-to-peer federated learning

**Question:** In sequential peer-to-peer (P2P) networks, how is reconstruction accuracy affected if the model is trained through multiple nodes?

### 4.5.1 Background

Our previous experiments focused on a centralized federated learning setup. To investigate the effectiveness of inversion attacks in a more decentralized setting, we implemented a P2P FL system. This system operates by sequentially sharing model weights between nodes in the network, allowing us to analyze how such attacks exploit such P2P FL systems.

#### Participating nodes

Our experiments adopt a distinctive approach by focusing on closely related nodes, aligning with real-world use cases such as healthcare, finance, or insurance fraud detection. Unlike traditional Federated Learning (FL) scenarios characterized by an extensive network of millions of nodes, our setup involves only a few closely related nodes, typically interconnected within the same business domain. This arrangement is designed to mirror practical considerations, where entities within a specific business domain contribute collaboratively to model training.

#### Optimization algorithm

As discussed in the background section (2.3), the FedAvg algorithm takes center stage in our research because it aligns with the specific characteristics of our sequentially arranged nodes in the P2P FL system. In a sequential federated learning setup, we share weights to update the global model, as our target algorithm is FedAvg. Also, weight sharing has some advantages to our sequential peer-to-peer system, as follows:

1. **Smoother Transfer:** Sharing weights allows each node to directly inherit the knowledge accumulated by the previous node(s) in the sequence. This eliminates the need for multiple rounds of gradient exchange, leading to smoother model updates for subsequent nodes.

2. **Continuity in Learning:** As weights are shared, each node in the se-

quence builds upon the model's knowledge acquired by its predecessors. This continuity in learning is essential for the model to capture underlying patterns and relationships within the data.

3. **Consistency in Model Representation:** Sharing weights ensures that all nodes in the sequence work with a consistent model representation. This consistency is important for maintaining a unified global model, even as the training progresses through different devices.

### 4.5.2 Implementation

#### Nodes setup:

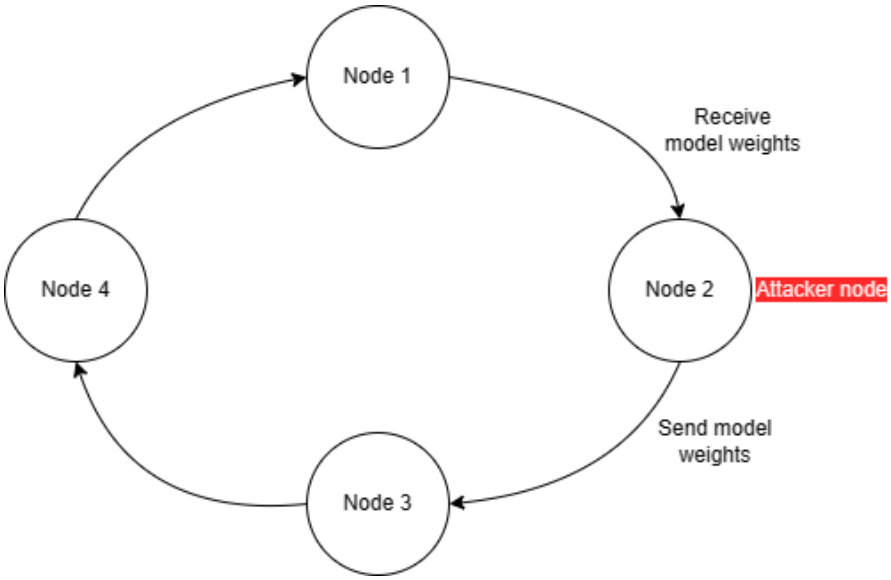


**Figure 4.4:** Architecture of peer-to-peer federated learning system

In our design of a sequential P2P FL system with 4 nodes (Figure 4.4), each participating node begins with identical initial copies of the model. The progression of updates within this system follows a notable pattern. Beginning with node n, it receives model updates from node n - 1. However, node n adopts a different approach rather than merely averaging these updates. It replaces its current model weights with the received updates and then trains on its respective dataset using this updated model locally. Following this local training, node n transmits the newly updated weights to node n + 1, thereby perpetuating the sequential chain of updates within the FL system. This iterative process ensures that each node in the system continuously incorporates the latest model updates from its predecessor while contributing its own refined model parameters to the subsequent nodes in the chain.

### Leakage attack

This implementation simulates a federated learning scenario with multiple nodes, each with its own dataset. In each round of training, one node is selected to be attacked. During the attack, the selected node uses an adversarial technique called federated averaging attack $fed\_avg\_attack$ to manipulate its model parameters. This attack aims to reconstruct the private training data of the previous nodes in the training chain. The reconstruction accuracy and the accuracy of the model are evaluated after each round of training to assess the impact of the attack on the overall performance. This process is repeated for a predefined number of rounds, with the results recorded for further analysis.

We encounter a unique challenge when working with a sequential P2P FL. Traditional FedAvg uses a central server to gather and combine model updates from all participating devices. Each device will give one update (weights), and the attacker node can do the inversion attack and should be able to recover the dataset used on that device for training. However, in a sequential P2P network with n nodes where updates flow sequentially, the attacker node (one of the n nodes) receives aggregated weights of all nodes since the last round as a single accumulated update, which is (n - 1) updates in total.

For example, let us say we have four nodes, and each node trains with 8 records. If Node 2 is the attacker node, it will receive updates from Node 3, Node 4, and Node 1. This means that the weights shared by the previous node (Node 1) have updates of 24 records in total.

To address this distinction and adapt the attack methodology for our scenario, we need to modify the module in standard TabLeak's $fed\_avg\_attack$ attack.

TabLeak's $fed\_avg\_attack$ attack employs a well-established framework from [20] to conduct attacks on the peer-to-peer system where weights are exchanged, unlike traditional federated learning setups that often rely on FEDSGD, where gradients are exchanged.

In this implementation, several technical aspects contribute to the effectiveness of the federated averaging attack:

1. **Reconstruction shape**: The number of records we are trying to reconstruct is $(nodes - 1) \times$ batch_size. The shape of the ground truth tensor, which is passed to $fed\_avg\_attack$, must be changed according to this. The algorithm will use this tensor shape to reconstruct the projected batch.

2. **Post-selection**: After each attack iteration, a post-selection process is employed to ensure the stability of the attack. This involves selecting the best reconstruction results from multiple candidate reconstructions, enhancing the reliability of the attack.

3. **Batch size**: A uniform batch size 16 is maintained across all nodes during training. This consistency ensures standardized processing and communication between nodes, facilitating the smooth execution of the federated learning algorithm.

4. **Best loss selection**: The attack iteratively refines its parameters to minimize the loss function associated with the reconstruction task. The best-performing reconstruction results, determined by the lowest loss values, are selected for further analysis and evaluation.

### 4.5.3 Result

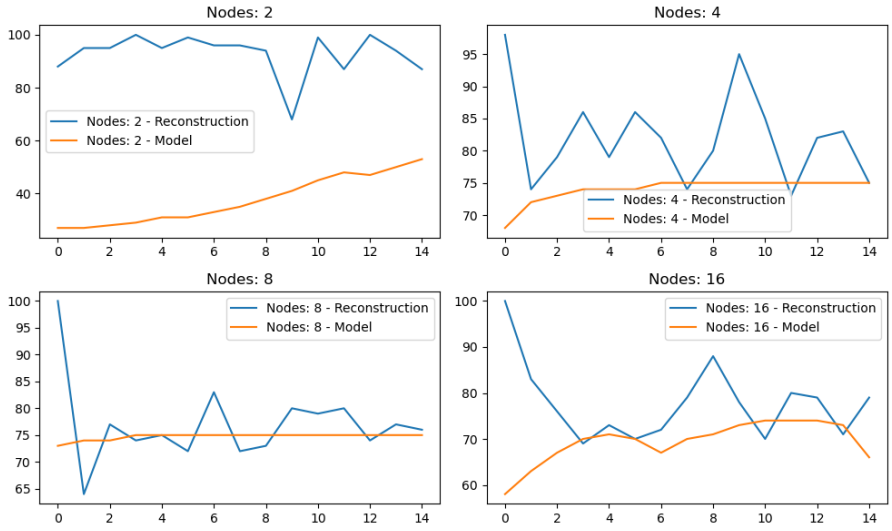


**Figure 4.5:** Reconstruction accuracy for sequentially connected nodes

This experiment evaluated the tableak performance of a P2P FL system across varying numbers of nodes. We observed that as the number of nodes increased, the reconstruction accuracy exhibited fluctuations, indicating the influence of the network topology on data reconstruction. Specifically, we achieved high reconstruction accuracy with two nodes, as it doesn't have enough updates. However, as the number of nodes increased to four, eight, and sixteen, the

* https://github.com/leopragi/tableak_fl/commit/sequential_p2p

reconstruction accuracy decreased, suggesting challenges associated with a larger set of participants.

Interestingly, while the model accuracy remained relatively stable across different node configurations, indicating robustness in learning performance, the reconstruction accuracy varied significantly.

### 4.5.4 Discussion and future work

In a sequentially connected set of nodes, the attacker node can recover certain training data points. However, the anonymity of the association between nodes and specific data points remains preserved. The recovery process lacks any discernible order, preventing the inference of node-to-data point relationships. However, the scalability of this approach for more extensive networks remains a concern. The iterative process of waiting for an update from the previous node is repeated until the model converges, which can be more time-consuming than centralized FL due to the sequential nature of updates. Although this setup may exhibit a slower convergence rate, we focus on it as it mirrors the real-world use case where clients often conduct local training and aggregation using a process similar to federated averaging. By addressing this scenario, we aim to provide insights and strategies for optimizing FL in such decentralized systems, acknowledging the potential trade-offs between convergence speed and data privacy.

Khac-Hoang et al. [28] examined SecAgg as a method for anonymizing individual updates and investigated its effectiveness in concealing information as the number of participants increases. SecAgg achieves anonymity by aggregating updates from multiple clients like our P2P FL. Their analysis revealed that in SecAgg, the server might still be able to infer the sources of updates based on the combined information, especially with a small number of participants [28]. The study's main finding is that SecAgg fails to adequately protect privacy when the model size exceeds the number of clients, as there is insufficient masking in the updates. Understanding how this attack works could help us identify vulnerabilities in our sequential FL system.

One effective defense strategy for the *FGLA attack* is to update the model multiple times locally and aggregate these updated gradients before uploading them. Yang et al. [29] introduce FGLA, a fast gradient leak attack method that can reconstruct a batch of user training data from gradients in milliseconds. Furthermore, we find that defenses such as batch aggregation, gradient compression, and noise addition struggle to protect against FGLA attacks.

## 4.6 Model reconstruction attack

**Question:** Does the model trained on reconstructed data perform as well as the model trained on original data?

### 4.6.1 Background

While the previous experiment achieved high model accuracy in P2P FL, the lower reconstruction accuracy raises an important question: Does achieving a lower reconstruction accuracy mean we are safe? Even though the attacker might not be able to reconstruct the original training data perfectly, if the reconstructed data still retains the same patterns and statistical properties as the true data, it can still be problematic.

Not all information in the data is equally important for the model to function well. Even if an attacker can only perfectly reconstruct a small portion of the data, it might be enough risk if that portion contains the fundamental knowledge the model uses to make predictions. Reducing reconstruction accuracy alone might not be sufficient to guarantee complete privacy protection. It is essential to ensure that the reconstructed data does not contain enough information to reveal the underlying patterns and relationships in the original training data.

### 4.6.2 Implementation

Figure 4.6 shows overview of the reconstruction model attack. A new model is trained using the reconstructed data and then tested against a true, unseen test dataset. This will directly assess how well the attacker's reconstruction translates to real-world data on which the actual model is trained. Training the model on reconstructed data helps us understand how well the model generalizes the pattern of the original data from the reconstructed data.

This implementation compares the performance of two models. One original model is trained on the ground truth data, while the other is trained on data that an attacker has reconstructed. By comparing their performance on the same test set (ground truth), we can assess how well the attacker's reconstruction model performs on the task the original model was designed for.

The code reads two datasets from CSV files. One contains the original data used to train a model, and the other contains data reconstructed by an attacker. Both datasets are split into training and testing sets. "Model True" is trained using the original data. "Model Recreated" is trained using the reconstructed

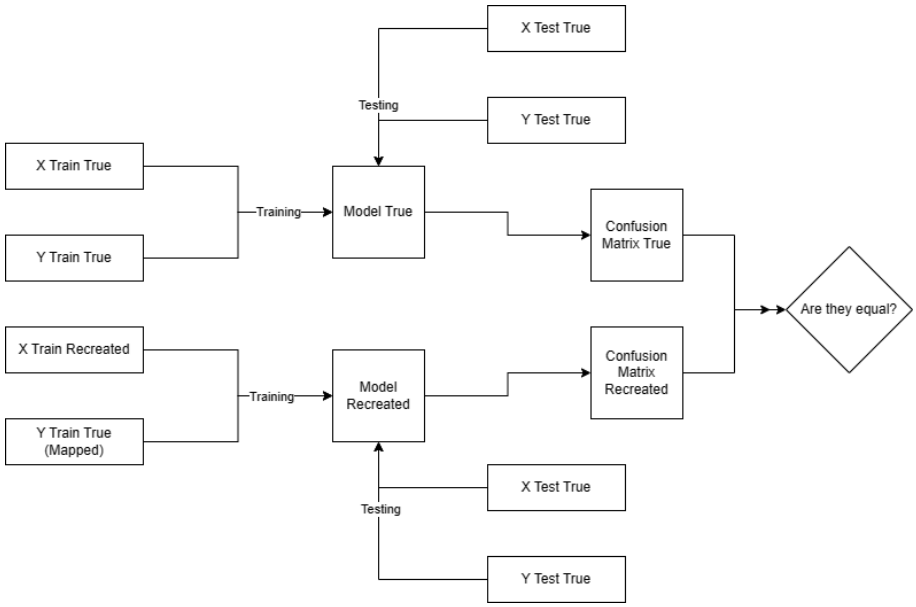


**Figure 4.6:** Model reconstruction attack overview

data provided by the attacker. Then, both of these models are tested with a true test data from the original data. Then, both models' performance are calculated based on their confusion matrix.

Along with this, we will see the distribution of the data reconstructed by the attacker and can compare it against the distribution of the original data. This can reveal potential biases or distortions introduced during the reconstruction process.

### 4.6.3 Result

The table 4.6 presents performance metrics of original and recreated models. Average reconstruction accuracy of the entire dataset is 58% which means recreated data is lower quality. The original model achieved an excellent overall model accuracy of 88.05%, and the recreated model has 75.45% model accuracy. Even though the recreated model seems to have good model accuracy, we need to analyze other metrics to get the full picture.

While the high recall on the reconstructed model might appear favorable, it is misleading due to the very low specificity. This suggests that the attacker's reconstructed data makes the model struggle to differentiate between positive

* https://github.com/leopragi/tableak_fl/commit/reconstructed_model

and negative cases.

| Metric | Original Model | Reconstructed Model |
|---|---|---|
| Confusion Matrix | $\begin{bmatrix} 885 & 401 \\ 229 & 3756 \end{bmatrix}$ | $\begin{bmatrix} 1 & 1285 \\ 9 & 3976 \end{bmatrix}$ |
| Accuracy | 88.05% | 75.45% |
| Precision | 90.35% | 75.57% |
| Recall (Sensitivity) | 94.25% | 99.77% |
| Specificity | 68.82% | 0.08% |
| F1 Score | 92.26% | 86.00% |

**Table 4.6:** Comparison of original and reconstructed models

Our examination of the data distribution across original and reconstructed data highlights significant discrepancies. This analysis focused on four different features that are: age (numeric), fnlgwt (numeric), education (categorical) and occupation (categorical). These results are present in figures 4.7, 4.8, 4.9 and 4.10. These features are affected by the reconstruction process as follows:

1. **Age:** The ground truth data shows a clear trend, with the positive class declining and the negative class increasing with age. However, both classes appear flat in the reconstructed data, with values extending beyond the original data range.

2. **fnlwgt:** In ground truth, both positive and negative classes follow a natural distribution of values. While the reconstructed data distribution becomes flat, the values remain within the expected range.

3. **Education:** The ground truth data indicates that 'HS-grad' is the most frequent category, followed by 'some-college' and 'bachelor.' However, in the reconstructed data, the 'HS-grad' category is entirely absent, indicating a significant alteration due to the reconstruction process. Despite this, 'Some-college' and 'bachelors' remain the top frequent values.

4. **Occupation:** In the ground truth, the most frequent categories are 'Adm-clerical' and 'Other-service occupation.' However, in the reconstructed data, the distribution is significantly altered, with 'Craft-repair,' 'Prof-specialty,' and 'Exec-managerial' now emerging as the top categories.

These observations suggest that the reconstruction process significantly altered the underlying distribution of the data, potentially introducing biases or distortions. This misalignment between the original data and reconstructed data distributions confirms that pattern in original data is not leaked via the reconstructed data. This result clearly shows that an attack cannot generalize

the pattern from the reconstructed data.

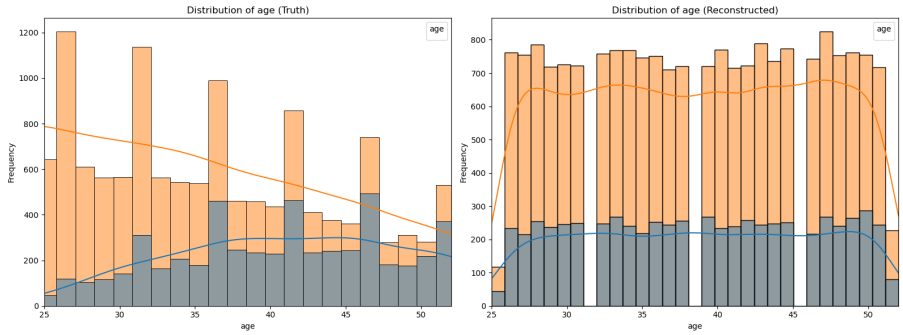


**Figure 4.7:** Distribution based on age

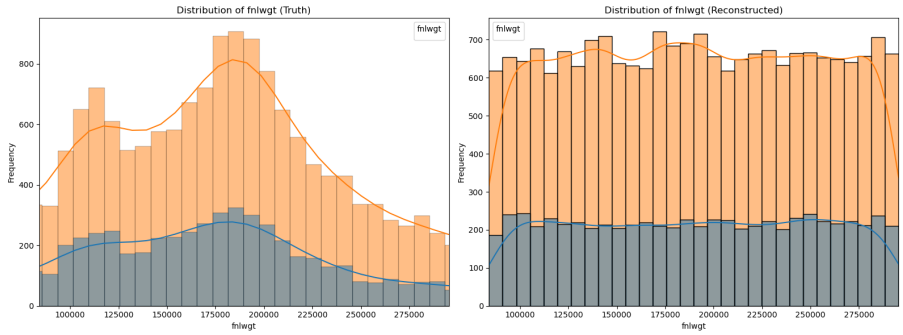


**Figure 4.8:** Distribution based on fnlwgt

### 4.6.4 Discussion and future work

Joshua et al. [30] focus on whether leaked datasets through reconstruction attacks (like gradient inversion and linear layer leakage) can still be used to train effective models, even with imperfect reconstructions. Gradient inversion attacks often suffer from lower-quality reconstructions. Quoting the research, "we also see a negative relationship between the increasing batch size and the usefulness of the leaked data in downstream model training" [30] aligns with our work. While this reduces the usefulness of the leaked data, the research shows it can still be used for training with some performance loss.

Our experiment revealed significant distribution discrepancies between the original and reconstructed data. A deeper understanding of how reconstruction processes introduce these biases and distortions can be valuable for developing more robust leakage detection methods.

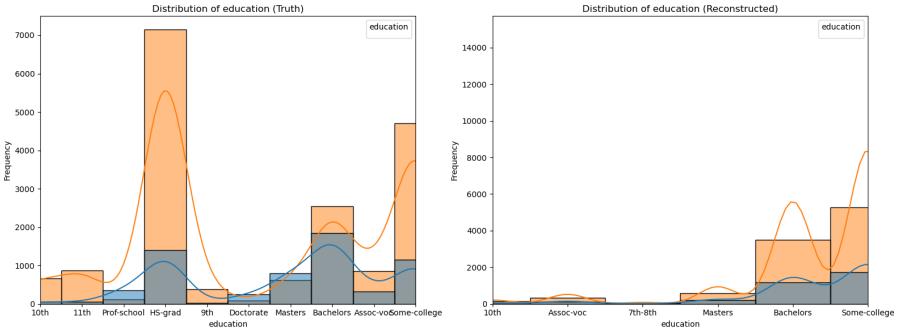


**Figure 4.9:** Distribution based on education

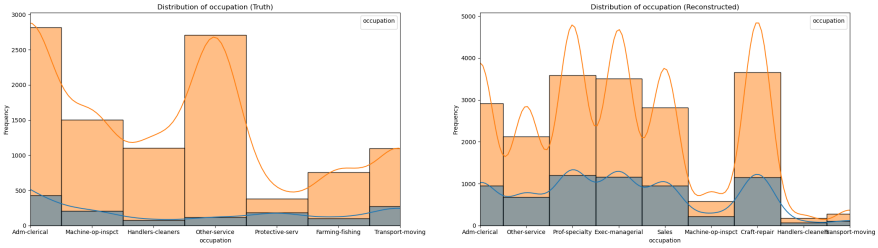


**Figure 4.10:** Distribution based on occupation

# 5 Conclusion

In this section, we will explore how to advance this research further and conclude this thesis.

## 5.1 Future works

Our current research focused on binary classification tasks within federated learning. However, a promising future direction would be to examine the impact of using ordinal output in the neural network (e.g., low, medium, high risk). Binary output simplifies the problem but might offer leakage attack a more straightforward target for reconstruction. The ordinal output provides additional information that can make it more difficult for attackers to pinpoint the exact original value. This exploration, particularly for datasets with naturally ordered output variables (like insurance fraud risk levels), could be a valuable step toward developing more robust and privacy-preserving FL systems.

Shaltiel et al. [31] suggest that changing the loss function from CrossEntropy to Mean Squared Error (MSE) might enhance privacy due to the "gradient mixing" properties of MSE. Problems with ordinal classification tasks can be treated as regression problems, which use the MSE loss function. Understanding the relationship between loss function and reconstruction accuracy can be crucial for developing a more comprehensive approach to privacy in federated learning. This future step proposes to explore whether this change in loss function also

impacts the reconstruction accuracy of attacks.

We have concluded even if perfect reconstruction isn't achieved, attackers might still get valuable information from partially reconstructed data. The 0-1 loss metrics don't account for the potential harm caused by such partial reconstructions. We need more robust metrics to assess reconstruction accuracy beyond just "perfect" reconstructions. This could involve analyzing the distribution of reconstructed data and its similarity to the original data in various aspects.

Also, developing machine learning methods to identify high-risk gradients that might leak sensitive information during training could be a powerful pre-emptive defense strategy.

## 5.2 Concluding remarks

This thesis explored various defense mechanisms against leakage attacks by investigating the effectiveness of various machine learning parameters. We demonstrated how modifying these parameters within the FL training process impacts the attacker's ability to reconstruct user data.

We have expanded the parameter landscape by exploring beyond traditional parameters to examine the impact of data encoding (e.g., binary vs. one-hot) and network architectures (e.g., sequential peer-to-peer vs. centralized). When modifying these parameters, we addressed the potential trade-offs between privacy and model performance. For instance, increasing batch size might improve privacy but could also lead to slower training times or scalability issues in a peer-to-peer architecture. We also validated the metrics of reconstruction accuracy by examining the pattern in the data.

This work contributes to developing privacy-preserving FL frameworks by examining potential mitigation strategies. The insights gained from this research will empower researchers and developers to create more secure FL systems, fostering wider adoption of this technology and safeguarding user privacy.

# Bibliography


[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (A. Singh and J. Zhu, eds.), vol. 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282, PMLR, 20–22 Apr 2017.

[2] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[3] Y. Wen, J. Geiping, L. Fowl, M. Goldblum, and T. Goldstein, "Fishing for user data in large-batch federated learning via gradient magnification," 2022.

[4] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov, "See through gradients: Image batch recovery via gradinversion," 2021.

[5] J. Deng, Y. Wang, J. Li, C. Wang, C. Shang, H. Liu, S. Rajasekaran, and C. Ding, "TAG: Gradient attack on transformer-based language models," in *Findings of the Association for Computational Linguistics: EMNLP 2021* (M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, eds.), (Punta Cana, Dominican Republic), pp. 3600–3610, Association for Computational Linguistics, Nov. 2021.

[6] M. Balunović, D. I. Dimitrov, N. Jovanović, and M. Vechev, "Lamp: Extracting text from gradients with language model priors," 2022.

[7] S. Gupta, Y. Huang, Z. Zhong, T. Gao, K. Li, and D. Chen, "Recovering private text in federated learning of language models," 2022.

[8] M. Vero, M. Balunović, D. Dimitrov, and M. Vechev, "Data leakage in tabular federated learning," 10 2022.

[9] S. K. Lo, Q. Lu, L. Zhu, H. young Paik, X. Xu, and C. Wang, "Architectural patterns for the design of federated learning systems," 2021.

[10] J. Qi, Q. Zhou, L. Lei, and K. Zheng, "Federated reinforcement learning: techniques, applications, and open challenges," *Intelligence Robotics*, vol. 1, 10 2021.

[11] T. Wink and Z. Nochta, "An approach for peer-to-peer federated learning," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 150–157, 2021.

[12] A. Karras, C. Karras, K. C. Giotopoulos, D. Tsolis, K. Oikonomou, and S. Sioutas, "Peer to peer federated learning: Towards decentralized machine learning on edge devices," in *2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, pp. 1–9, 2022.

[13] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," 2019.

[14] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[15] N. Bouacida and P. Mohapatra, "Vulnerabilities in federated learning," *IEEE Access*, vol. 9, pp. 63229–63249, 2021.

[16] G. S. Dhillon, K. Azizzadenesheli, Z. C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, and A. Anandkumar, "Stochastic activation pruning for robust adversarial defense," 2018.

[17] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics* (S. Chiappa and R. Calandra, eds.), vol. 108 of *Proceedings of Machine Learning Research*, pp. 2938–2948, PMLR, 26–28 Aug 2020.

[18] M. Carminati, L. Santini, M. Polino, and S. Zanero, "Evasion attacks against banking fraud detection systems," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, (San Sebastian), pp. 285–300, USENIX Association, Oct. 2020.

[19] K.-C. Wang, Y. FU, K. Li, A. Khisti, R. Zemel, and A. Makhzani, "Variational model inversion attacks," in *Advances in Neural Information Processing Systems* (M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), vol. 34, pp. 9706–9719, Curran Associates, Inc., 2021.

[20] D. I. Dimitrov, M. Balunovic, N. Konstantinov, and M. Vechev, "Data leakage in federated averaging," *Transactions on Machine Learning Research*, 2022.

[21] W. Wei, L. Liu, M. Loper, K.-H. Chow, M. E. Gursoy, S. Truex, and Y. Wu, "A framework for evaluating gradient leakage attacks in federated learning," 2020.

[22] O. Goldreich, "Secure multi-party computation," *Manuscript. Preliminary Version*, 03 1999.

[23] X. Li, Y. Gu, N. Dvornek, L. H. Staib, P. Ventola, and J. S. Duncan, "Multi-site fmri analysis using privacy-preserving federated learning and domain adaptation: Abide results," *Medical Image Analysis*, vol. 65, p. 101765, 2020.

[24] B. Becker and R. Kohavi, "Adult." UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[25] F. L. Wightman, "LawSchool." LSAC national longitudinal bar passage study, 2017.

[26] S. G. K. Patro and K. K. Sahu, "Normalization: A preprocessing stage," 2015.

[27] C. Seger, "An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing," 2018.

[28] K.-H. Ngo, J. Östman, G. Durisi, and A. G. i Amat, "Secure aggregation is not private against membership inference attacks," 2024.

[29] H. Yang, D. Xue, M. Ge, J. Li, G. Xu, H. Li, and R. Lu, "Fast generation-based gradient leakage attacks: An approach to generate training data directly from the gradient," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–13, 2024.

[30] J. C. Zhao, A. Dabholkar, A. Sharma, and S. Bagchi, "Leak and learn: An attacker's cookbook to train using leaked data from federated learning," 2024.

[31] S. Eloul, F. Silavong, S. Kamthe, A. Georgiadis, and S. J. Moran, "Enhancing

privacy against inversion attacks in federated learning by using mixing gradients strategies," 2022.

# Appendices

# A

# External Resources

## Declaration of the usage of AI tools

I acknowledge the use of Gemini (https://gemini.google.com/) to refine my work's academic language and accuracy. I used a grammar and style checking tool - Grammarly (https://www.grammarly.com/), to improve my academic tone and accuracy of language, including grammatical structures, punctuation, and vocabulary. The output from the tools was then modified further better to represent my own tone and style of writing.

## Declaration of Code Usage

In this thesis, the open-source code from the "tableak" repository on GitHub (https://github.com/eth-sri/tableak) was utilized to conduct different reconstruction attacks against federated learning models. This code, developed by eth-sri, implements techniques for reconstructing tabular data from gradients as described in the research paper [8]. We are grateful to the authors for making this valuable code publicly available.