CHALMERS
UNIVERSITY OF TECHNOLOGY

# Reconstructing Private Data from Trained Models

## An Application of Model Inversion Attacks on Deep Neural Networks, with a Novel Focus on the Tabular Data Domain

Master's thesis in Engineering Mathematics and Computational Science

Oscar Wernqvist and Hevar Djeza

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

DEGREE PROJECT REPORT 2025

# Reconstructing Private Data from Trained Models

An Application of Model Inversion Attacks on Deep Neural Networks, with a Novel Focus on the Tabular Data Domain

Oscar Wernqvist and Hevar Djeza

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025

Reconstructing Private Data from Trained Deep Neural Networks
An Application of Model Inversion Attacks on Deep Neural Networks, with a Novel Focus on the Tabular Data Domain
OSCAR WERNQVIST, HEVAR DJEZA



Supervisor: Johan Östman & Fazeleh Hoseini, AI Sweden
Examiner: Johan Jonasson, Department of Mathematical Sciences

Degree project report 2025
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Reconstructing Private Data from Trained Deep Neural Networks
An Application of Model Inversion Attacks on Deep Neural Networks, with a Novel Focus on the Tabular Data Domain
Oscar Wernqvist and Hevar Djeza
Department of Mathematical Sciences
Chalmers University of Technology

## Abstract

This thesis investigates whether Model Inversion (MI) attacks can be effectively adapted to tabular data—a domain where risks are underexplored compared to the image modality. To address this question, we propose a novel adaptation of the Pseudo-Label Guided Model Inversion (PLG-MI) attack for tabular data by utilizing a Conditional Tabular Generative Adversarial Network (CTGAN). In support of this contribution, new evaluation metrics are proposed—most notably, class-level column shape scores—which serve to measure the similarity between reconstructed and original private data. These metrics offer a practical means to evaluate the privacy risks posed by inversion attacks in the tabular setting. As an initial step, we have reproduced the PLG-MI attack for images, and verified that the attack is robust even on target models trained on unbalanced private data sets. Then, by applying the adapted tabular attack to a deep neural network diagnosis classifier trained on the MIMIC-IV clinical dataset, we demonstrate that sensitive features can be recovered with high accuracy. This shows that MI attacks can generalize to the tabular domain; with default hyperparameters and minimal tuning, our method recovers sensitive features with high accuracy. We have identified that the use of transformations between structured and unstructured data, as well as the common use of tree-based models in the tabular domain, can prevent adversarial gradient access, thereby limiting the applicability of white-box model inversion attacks to specific scenarios. Overall, our results confirm that model inversion attacks pose a real privacy threat in the tabular domain while also clarifying the technical boundaries that define when such attacks are viable. Our work will be made available as a part of the LeakPro repository: github.com/aidotse/leakpro

Keywords: Model Inversion Attacks, Reconstruction Attacks, Adversarial Machine Learning.

# Acknowledgements

We want to acknowledge our AI Sweden supervisors, Fazeleh Hoseini and Johan Östman, for their support and guidance on this project. We also want to thank the LeakPro team in its entirety for providing the foundational codebase to build this project upon. Thank you also to Johan Jonasson for acting as academic supervisor and showing interest in this project.

Oscar Wernqvist and Hevar Djeza, Gothenburg, June 2025

# List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

| | |
|---|---|
| CDM | Conditional Diffusion Model |
| CDF | Cumulative Distribution Function |
| CE | Cross-entropy |
| cGAN | Conditional Generative Adversarial Network |
| CTGAN | Conditional Tabular Generative Adversarial Network |
| DNN | Deep Neural Network |
| FID | Fréchet inception distance |
| GAN | Generative Adversarial Network |
| GANDALF | Gated Adaptive Network for Deep Automated Learning of Features |
| GDPR | General Data Protection Regulation |
| IFG-MI | Intermediate Feature-enhanced Generative Model Inversion |
| i.i.d | Independent and Identically Distributed |
| KED-MI | Knowledge-enriched Distributional Model In- version Attacks |
| KSS | Kolmogorov-Smirnov Statistic |
| LOMMA | Logit Maximization + Model Augmentation |
| LOKT | Label-Only Knowledge Transfer Attack |
| MI | Model Inversion |
| MIA | Membership Inference Attack |
| MIMIC | Multi-parameter Intelligent Monitoring for Intensive Care |
| MM | Max-margin |
| OOD | Out-of-distribution |
| PII | Personal Identifiable Information |
| PLG-MI | Pseudo Label-Guided Model Inversion Attack |
| SDV | Synthetic Data Vault |
| SOTA | State of the art |

# Nomenclature

Below is the nomenclature of sets, parameters, variables and functions that have been used throughout this thesis.

## Sets

| | |
|---|---|
| $\mathcal{D}$ | Data distribution |
| $\mathcal{D}_{\text{priv}}$ | Private dataset |
| $\mathcal{D}_{\text{pseudo}}$ | Pseudo labeled dataset |
| $\mathcal{D}_{\text{pub}}$ | Public dataset |
| $\mathcal{D}_{\text{train}}$ | Training dataset |
| $\mathcal{X}$ | Input space |
| $\mathcal{Y}$ | Label space |
| $\mathcal{Z}$ | Latent space |

## Parameters

| | |
|---|---|
| $\eta$ | Learning rate |
| $\alpha$ / $\alpha_1$ | Inversion loss scaling parameter |
| $\alpha_2$ | Conditioning loss scaling parameter |

## Variables

| | |
|---|---|
| $\boldsymbol{a}^{(\ell)}$ | Neuron activation of layer $\ell$ |
| $D$ | Discriminator model |
| $\boldsymbol{\theta}_D$ | Discriminator model parameters |
| $E$ | Evaluation model |

| | |
|---|---|
| $\boldsymbol{\theta}_E$ | Evaluation model parameters |
| $G$ | Generator model |
| $\boldsymbol{\theta}_G$ | Generator model parameters |
| $T$ | Target model |
| $\boldsymbol{\theta}_T$ | Target model parameters |
| $\boldsymbol{x}$ | Sample |
| $y$ | Label |
| $y_{\text{pred}}$ | Label prediction |
| $\boldsymbol{z}$ | Latent space embedding vector |

# Functions

| | |
|---|---|
| $\mathcal{L}^{\text{MM}}$ | Max-marigin loss |
| $\mathcal{L}^{\text{CE}}$ | Cross-entropy loss |
| $\mathcal{L}_{\text{inv}}$ | Inversion loss |
| $\mathcal{L}_{\text{cond}}$ | Conditioning loss |
| $\mathcal{L}_G$ | Generator loss |
| $\mathcal{L}_D$ | Discriminator loss |
| $\varphi$ | Activation function |
| $f_G$ | Generator transformations |
| $f_T$ | Target model transformations |

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# 1
# Introduction

In recent years, machine learning models have achieved significant success in a wide range of sectors. Their capability to model complex patterns and relevant relationships in data has made them a crucial tool in various data analysis applications. Among the various types of machine learning models, Deep Neural Networks (DNNs), which are characterized by their vast amount of parameters and complex architectures, have been of significant interest due to their applicability on large, high-dimensional datasets and remarkable performance in tasks such as image recognition and natural language processing.

However, the additional complexity of these models introduces obscurity regarding the learning process and how these models incorporate and retain information from training data, raising concerns about the potential for sensitive data to be unintentionally memorized and subsequently extracted based on the model's behavior when processing input data [1]. This is particularly relevant when considering models that have trained on data containing confidential or Personal Identifiable Information (PPI), potentially breaking confidentiality laws and regulations such as the General Data Protection Regulation (GDPR) [2].

A privacy attack refers to any method where an adversary, based on internal representations or output, aims to extract private training data from a trained model; often called the target model in literature. Among the various types of attacks, two prominent attacks are Membership Inference Attacks (MIAs) and Model Inversion reconstruction attacks. In the first attack, an adversary infers whether or not data points drawn from the same distribution as the training data were used to train the target model, while in the second attack the adversary aims to reconstruct private training data. The inference an adversary can make on the training data from a successful MI attack is thereby particularly strong, but is in general a more ill-posed task than MIAs [3].

Most model inversion reconstruction attacks have focused on image data; especially in the context of face recognition; leaving it unclear whether such attacks generalize to other data modalities. To the best of our knowledge, there exists no prior research that investigates the applicability of MI reconstruction attacks on tabular data. This thesis aims to address this gap in research by evaluating whether a current state-of-the-art MI reconstruction attack, namely the Pseudo Label-Guided Model Inversion Attack [4], can be effectively applied in the tabular domain. By exploring this unexamined setting, we contribute to a more comprehensive understanding of the robustness and limitations of MI reconstruction attacks across data modalities.

## 1.1 Purpose

Model Inversion attacks have emerged as a critical tool for evaluating the privacy vulnerabilities of machine learning models, particularly DNNs. These attacks aim to reconstruct sensitive attributes or entire data samples from model outputs, posing significant risks when models are trained on personal data.

The implications of MI attacks are especially clear in the context of data protection regulations such as GDPR. Under GDPR, if a model can be exploited to reveal PII from its training data, it may be considered as processing personal data itself, thereby subjecting it to strict compliance requirements [5]. This underscores the necessity for auditing methods like MI attacks to assess and mitigate potential privacy breaches before model deployment.

Beyond the privacy risks, MI attacks offer valuable insights into how DDNs learn and store data. By analyzing which data is vulnerable to MI reconstruction, researchers can identify patterns that lead to unintended memorization of training data, which is often a byproduct of overfitting. Understanding these patterns can inform the development of training techniques and model architectures that are more robust to privacy attacks.

While MI attacks have been studied in the image domain, their application to tabular data remains underexplored. Tabular datasets, common in sectors such as healthcare, finance, social sciences and many others, present unique challenges due to their mixed data types and complex feature relationships. Investigating MI attacks in this domain is crucial, as models trained on tabular data are equally susceptible to privacy risks, and the lack of research leaves a gap in our understanding of these vulnerabilities [6].

Therefore, our work aims to extend the study of MI attacks to tabular data, evaluating the robustness of existing attack methods in this context. This research is essential for developing comprehensive privacy-preserving strategies across all applications of machine learning models.

In order to understand and adapt MI attacks to tabular data, we first need to implement an existing attack on images. The purpose of this is to understand and analyze what aspects of an attack, the choice of dataset, target model and attack methodology, affect the outcome of the reconstruction. Conclusions from the image domain can then be applied to guide the creation of a successful attack on tabular data.

## 1.2 Goals

The goal of this thesis is to analyze and evaluate the efficacy of state-of-the-art model inversion attacks and propose ways to adapt these methods from the image domain to the tabular domain. By doing this we want to answer the following research questions:

- What are the key concepts utilized in state-of-the-art model inversion attacks and what are the limitations of these methods?
- How can these methods be adapted to attack models in the tabular data domain? Can we develop the first adaptation to tabular data of a modern model inversion reconstruction attack?
- The evaluation metrics used are not robust across domains [4], so in the field, there is no unified metric to rely on for an objective assessment of privacy. These metrics also do not accurately assess contextual information, such as how the identity of a person in an image can be identified. How can the implications of model inversion attacks be evaluated in the tabular domain, and what metrics are suitable for that evaluation?

# 2

# Background

Model Inversion attacks aim to reconstruct attributes or full data points used in the training of a model. This concept was first introduced by Fredrikson et al. in 2014 for attacking a linear regression target model predicting drug dosage based on sensitive patient attributes [7]. In 2015, Fredrikson et al. developed a MI attack on a Neural Network trained on images of faces [8]. This paper has inspired multiple subsequent works and defined one of the main applications of MI attacks.

## 2.1 Threat Model

In an MI attack (see Algorithm 1), an adversary (the attacker) aims to recover private training data from a target model $T : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space. Here, $T$ is trained on a private data set $\mathcal{D}_{\text{priv}}$, and the adversary has a level of access $A(T(\cdot))$ to the target model. In addition to this, there exists a public dataset $\mathcal{D}_{\text{pub}} \subset \mathcal{X}$ such that the adversary can use samples $\boldsymbol{x} \in \mathcal{D}_{\text{pub}}$ as input to $T$. The goal of the adversary is then to find members of $\mathcal{D}_{\text{priv}}$ by creating samples $\tilde{\boldsymbol{x}}$ theoretically minimizing some distance metric $d$,

$$\tilde{\boldsymbol{x}} = \{\min_{\tilde{\boldsymbol{x}}} d(\tilde{\boldsymbol{x}}, \boldsymbol{x}) : \boldsymbol{x} \in \mathcal{D}_{\text{priv}}, \tilde{\boldsymbol{x}} \in \mathcal{X}\}. \tag{2.1}$$

However, because $\mathcal{D}_{\text{priv}}$ is not available to the adversary, this instead has to be approximated in other ways by extracting knowledge available by $A(T(\cdot))$, which can be predictions, gradients or internal representations, to gain insights about the training data set $\mathcal{D}_{\text{priv}}$.

**Algorithm 1** Model Inversion Attack

**Input:** Target model $T$ with access level $A$, public dataset $\mathcal{D}_{pub}$

1: Extract knowledge available from $A(T(\boldsymbol{x}))$, $\boldsymbol{x} \in \mathcal{D}_{\text{pub}}$.
2: Define a generative reconstruction strategy on how to generate candidate samples $\tilde{\boldsymbol{x}}$ conditioned on the knowledge from $T$.
3: Reconstruct synthetic samples: Generate candidate samples $\tilde{\boldsymbol{x}}$ and optimize them to better match the learned distribution of $T$.

**Output:** Reconstructed data samples $\tilde{\boldsymbol{x}}$

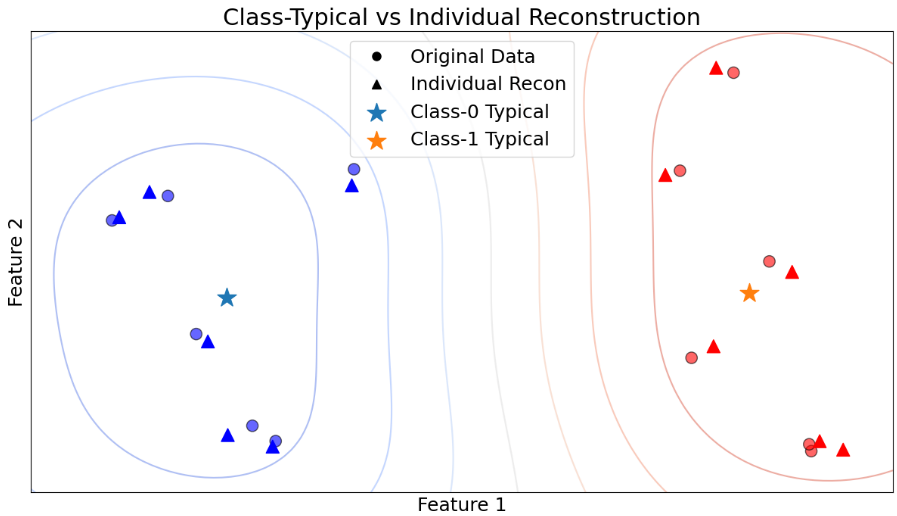



**Figure 2.1:** Illustrative example of differences between class typical and individual reconstruction. The contour lines represent class likelihood. The triangles represent points that an individual reconstruction strategy could find, and the stars represent points that a class typical reconstruction strategy could find.

### 2.1.1 Black and White-box attacks

In a Black-Box attack, the adversary only has access to the output of the model, $A(T(\boldsymbol{x})) \in \mathcal{Y}$, which takes the shape of a confidence vector. A special case of a Black-Box attack is a Label-only attack, where the only information available to the adversary is the predicted label from the target model. Due to the restricted information available to the adversary in Black-Box attacks, they are generally weaker attacks compared to White-Box attacks.

In a White-Box attack, the adversary has access to additional information about the target model behavior. This is usually complete information about the gradients and parameters of the model, $A(T(\boldsymbol{x})) \in \{T(\boldsymbol{x}), \frac{\partial}{\partial \boldsymbol{x}} T(\boldsymbol{x}), \boldsymbol{\theta}_T\}$. This additional information allows for the possibility of a stronger attack.

### 2.1.2 Typical vs. individual reconstruction

The most well-known types of model inversion attacks are *typical* reconstruction attacks. This means that they aim to reconstruct class-typical samples $\tilde{\boldsymbol{x}}$ likely to be from the same distribution as training samples $\boldsymbol{x}$ belonging to a specific target class $y$. The other type of model inversion attacks are *individual* reconstruction attacks, where the objective is to reconstruct samples $\tilde{\boldsymbol{x}}$ corresponding individual training samples $\boldsymbol{x}$ directly. The differences of this can be illustrated by Figure 2.1. Here, the blue and red corresponds to two different classes, and the contour lines indicate the class likelihood as seen by the target model. Individual reconstruction would produce the triangles, samples close to original data points. Typical reconstruction would instead produce some point with high likelihood of being within the class, represented by the star shapes.

Typical reconstruction attacks are by nature only applicable in the setting where the target model is a classifier such that $\arg\max T(\boldsymbol{x}) = y_{pred}$, in contrast to individual reconstruction, which for example has been applied to regression models [7]. Class-typical reconstruction also implies that the attacks only reveal usable information about the training data if the class distributions themselves contain distinct and meaningful information.

As an example, if the target model is an image classifier distinguishing cats from dogs, the typical reconstruction model inversion attack would aim to reconstruct samples $\tilde{\boldsymbol{x}}$ corresponding to the cat class or the dog class. In this case, the reconstructed samples only show how a typical cat or dog from the training set looks like, which has limited usability to the adversary.

In contrast, a more relevant example of class-typical reconstruction attacks is in facial recognition systems. Suppose a model is trained to classify individuals based on their facial images, with each person being a separate class in the classifier. A class-typical reconstruction attack would attempt to generate a synthetic face that represents the general appearance of a specific individual from the training set. This can in the right context be considered a breach of privacy.

With this in mind, individual reconstruction attacks pose a theoretical privacy threat even in cases where class-typical reconstruction is uninformative. However, under the constraints of the threat model defined in Section 2.1, and to the best of our knowledge, only class-typical reconstruction attacks have been demonstrated to be feasible against DNNs in practice [9].

### 2.1.3 Model Inversion Attacks in the Image Domain

The existing literature on Model Inversion Attacks focuses on reconstructing the private data from a trained face recognition classifier. To this end, some prior knowledge is assumed, called a public dataset. This dataset is in some papers assumed to be drawn from the same distribution as the private dataset, and in some papers drawn from a distribution with some distributional shift.

The attack and reconstruction is often performed with a Generative Adversarial Network (GAN). The GAN is trained to be able to accurately generate images that are likely to belong to either the private dataset or a specific class within that dataset.

As discussed previously, facial recognition tasks are the most common application of model inversion attacks in the image domain. Beyond this use case, few other examples exist in the literature where class-typical image reconstruction would be considered a breach of privacy. Because of this, we will also look at the tabular domain.

## 2.2 Limitations / Demarcations

In this paper, we will generally evaluate privacy on a worst-case basis. In practice, this means strong assumptions on the adversary's capabilities;

- The adversary has access to public data from the same distribution as the private data, $\mathcal{D}_{\text{priv}} \sim \mathcal{D}_{\text{pub}}$. In practice, this means sampling $\mathcal{D}_{\text{priv}}$ and $\mathcal{D}_{\text{pub}}$ from the same data set. This is referred to as the in-distribution case, in contrast to the out-of-distribution case where they are sampled from different data sets.
- The adversary has access to all information about the target model, including gradients if they exist. This means that we will focus on white-box attacks.

The purpose of these assumptions is to determine what is theoretically possible for an adversary to achieve, rather than what is practically feasible. If a target model is robust against the strongest assumed adversary, it is also expected to withstand attacks from less capable adversaries.

Moreover, by setting the strongest adversarial capabilities, we establish optimal conditions for successful attacks. This is essential to answer whether MI reconstruction attacks are feasible on tabular data, a necessary step in identifying how the attacks can be adapted to the tabular data domain.

We will explore typical reconstruction attacks due to the existence of previous literature on the topic. We leave individual reconstruction attacks in our setup for future work.

## 2.3 Related Works

In this section, we introduce related model inversion attacks along with their capabilities and limitations, with consideration to their cross-domain applicability. All of these works are focused on typical model inversion reconstruction attacks on facial classification DNNs.

### 2.3.1 The KED-MI Attack

Some of the foundational principles of recent Model Inversion Attacks were established by Chen et al. in the paper *Knowledge-enriched Distributional Model Inversion Attacks* [10]. The notion of a training phase and an attack phase of model inversion attacks is clearly defined in this paper. They utilize a GAN as their generative model, with a custom conditional discriminator that incorporates target model predictions during the training phase. In the attack phase, the goal is to optimize the latent space used as input for the GAN. They do this by finding Monte Carlo estimates of some Gaussian distribution, using information from both the target model and the discriminator as guidance.

### 2.3.2 The PLG-MI Attack

The paper *Pseudo Label-Guided Model Inversion Attack via Conditional Generative Adversarial Network* (PLG-MI) by Yuan et al. [4] introduces a conditional GAN framework guided by pseudo-labels to optimize the reconstruction process. By using a top-n strategy to assign pseudo-labels to public data, PLG-MI narrows the optimization space for each class, making it easier for the GAN to generate class-relevant samples. It also proposes a max-margin loss to boost intra-class fidelity. This method shows state-of-the-art performance and robustness across varied distribution shifts and importantly, does not require a pre-trained GAN, making it applicable for tasks where pre-trained GANs do not exist. Similarly to KED-MI, PLG-MI consists of a GAN-training phase, and an attack phase to optimize the reconstructed samples.

### 2.3.3 The IF-GMI Attack

Intermediate Feature-enhanced Generative Model Inversion (IF-GMI) [11] enhances the pseudo-labeling method of PLG-MI by exposing and optimizing the intermediate features of the GAN. This expands the search space beyond latent vectors, improving the granularity of reconstructions. An $\ell_1$-ball constraint ensures realistic outputs while maintaining fidelity. IF-GMI is shown to be effective in both in-distribution and out-of-distribution image settings, though it relies on a pre-trained generator. This limits the applicability of IFG-MI to settings where there exists a suitable pre-trained generator.

### 2.3.4 The LOMMA Attack

In Re-thinking Model Inversion Attacks Against Deep Neural Networks, Nguyen et al. introduces the Logit Maximization + Model Augmentation Attack (LOMMA) [12], revisiting the optimization objectives of previous attacks. It introduces a logit loss to preserve identity features and employs model augmentation to regularize the optimization. These enhancements lead to improved reconstructions without modifying the target model, though the framework is designed primarily for the image domain.

### 2.3.5 The LOKT Attack

The Label-Only Knowledge Transfer Attack (LOKT) [13] targets scenarios where only the hard labels (top-1 class outputs) of a target model are available. To overcome the lack of gradients, LOKT proposes T-ACGAN to learn from the limited label information by transferring knowledge to surrogate models. This surrogate-based approach offers flexibility, particularly when direct access to the model is restricted. While LOKT is shown to be comparatively strong in the black-box setting, the paper also demonstrates that it remains weaker than white-box alternatives in the standard facial recognition benchmarks.

### 2.3.6 Comparative Analysis

A comprehensive comparison of model inversion attacks is presented in MIBench [14], which benchmarks 16 state-of-the-art Model Inversion Attacks in various settings on 4 facial recognition data sets. It highlights key limitations in scalability and generalization. Notably, MIBench underscores the dependency of most generative attacks (such as PPA, IF-GMI, and LOKT) on pretrained GANs or synthetic data generators, restricting their applicability to domains like tabular data where such generators are not easily available.

One of the key results that MIBench shows is the strong positive correlation between the predictive power of the target model and the effectiveness of model inversion attacks. They demonstrate this relationship by evaluating attack accuracy against a ResNet-152 model trained to varying levels of predictive power. Their findings indicate that models with higher predictive accuracy are more vulnerable to inversion attacks. This insight is particularly relevant for our work in the tabular data domain.

For our task, which explores model inversion on tabular data, we require a method that does not rely on pretrained generative models. Additionally, to understand the inherent risk under ideal adversarial conditions, we want to focus on the white-box setting to allow for the strongest possible attack when investigating a new domain. PLG-MI meets both criteria: it avoids the need for pretrained GANs by using public data and pseudo-labels, and according to MIBench, PLG-MI also offers competitive or superior performance compared to the other GAN-based white-box attacks, making it the most suitable approach for our objectives.

# 3

# Theory

## 3.1 Artificial Neural Network Architecture

Artificial neural networks are machine learning models consisting of layers of artificial neurons. These layers are the input layer, set by the input of the model, hidden layers, and the output layer corresponding to the output of the model. Neural networks with multiple hidden layers are referred to as Deep neural networks.

### 3.1.1 Forward Propagation

In order for a neural network to make a prediction $y_{pred}$ based on some input $\boldsymbol{x}$, that input needs to propagate through the neurons of the network. First, the input activation is set as $\boldsymbol{a}^{(0)} = \boldsymbol{x}$, and then appending it with a constant to incorporate the bias term:

$$\boldsymbol{a}^{(0)} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}.$$

At each layer of the network $\ell = 1, 2, \ldots, L$, we compute a pre-activation vector $\boldsymbol{z}^{(\ell)}$ by applying the affine transformation given by the parameters $\boldsymbol{\theta}^{(\ell)}$. Collecting weights and biases into a single object, we simply write

$$\boldsymbol{z}^{(\ell)} = \boldsymbol{\theta}^{(\ell)} \boldsymbol{a}^{(\ell-1)},$$

and then apply some nonlinear activation function $\varphi$ element-wise to obtain the next activation of the subsequent neuron layer

$$\boldsymbol{a}^{(\ell)} = \varphi\Big(\boldsymbol{z}^{(\ell)}\Big) = \varphi\Big(\boldsymbol{\theta}^{(\ell)} \boldsymbol{a}^{(\ell-1)}\Big).$$

Repeated substitution shows that after $L$ such layers, the network's prediction can be written as the nested composition

$$y_{pred} = \boldsymbol{a}^{(L)} = \varphi\Big(\boldsymbol{\theta}^{(L)} \varphi\Big(\boldsymbol{\theta}^{(L-1)} \cdots \varphi\Big(\boldsymbol{\theta}^{(1)} \boldsymbol{x}\Big) \cdots\Big)\Big).$$

Thus, forward propagation implements the mapping

$$T(\boldsymbol{x}; \boldsymbol{\theta}) = \Big(\varphi \circ \boldsymbol{\theta}^{(L)} \circ \cdots \circ \varphi \circ \boldsymbol{\theta}^{(1)}\Big)(\boldsymbol{x}) := y_{pred}.$$

where $y_{pred}$ is the network prediction, also known as the logit output. See chapter 5 in Mehlig [15] for a more detailed description of neural network dynamics.

### 3.1.2 Backpropagation

In order for a neural network to learn, the parameters of the network need to be adjusted according to some training dataset $\mathcal{D}_{train}$.

This is done by iterative updates of the model parameters $\boldsymbol{\theta}_T$ by feeding samples $(\boldsymbol{x}, y) \in \mathcal{D}_{train}$ into the model and comparing the model prediction $T(\boldsymbol{x}) = y_{pred}$ to the target value $y$.

Backpropagation involves calculating the gradient of some loss function $\mathcal{L}$ with respect to the model parameters $\boldsymbol{\theta}_T$. If the loss is computed over a single training sample, then the parameters $\boldsymbol{\theta}_T$ are updated with a learning rate $\eta$ according to the learning rule:

$$\boldsymbol{\theta}_T \leftarrow \boldsymbol{\theta}_T - \eta \frac{\partial}{\partial \boldsymbol{\theta}_T} \mathcal{L}(y, y_{pred}). \tag{3.1}$$

### 3.1.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) applies the single-sample update rule iteratively over the dataset. Rather than computing the gradient of the average loss over the entire training set at each step, SGD updates the parameters using only a single (or a small batch of) training example(s) at a time. Concretely, given one training pair $(\boldsymbol{x}, y) \in \mathcal{D}_{\text{train}}$, we first compute the prediction

$$y_{pred} = T(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{a}^{(L)} = \varphi\Big(\boldsymbol{\theta}^{(L)} \varphi(\boldsymbol{\theta}^{(L-1)} \cdots \varphi(\boldsymbol{\theta}^{(1)} \boldsymbol{x}) \cdots )\Big),$$

and then the corresponding sample loss $\mathcal{L}(y, y_{pred})$. The parameters are then updated according to the learning rule in Equation 3.1. After this, a new training pair $(\boldsymbol{x}, y)$ is sampled.

By repeating this procedure, the parameters $\boldsymbol{\theta}$ are driven toward values that (approximately) minimize the overall loss. This noisy, sample-by-sample approach often converges faster in practice than full-batch gradient descent, and introduces stochasticity that can help escape shallow local minima.

### 3.1.4 Adam

The *Adam* optimization algorithm was introduced by Kingma and Ba in 2015 [16]. It improves upon Stochastic Gradient Descent by addressing key issues such as:

- Adaptive Learning Rates: Adam computes individual learning rates for each parameter by utilizing the moment (mean and variance) estimates of the gradients.
- Momentum-Based Updates: It builds upon ideas from Momentum-based SGD, leading to more stable updates.
- Bias Correction: Adam corrects bias in the moving averages, especially in early training stages.

Concretely, at step $t$ we first compute the network's prediction

$$y_{\text{pred}} = f(\boldsymbol{x}; \boldsymbol{\theta}) = \varphi\Big(\boldsymbol{\theta}^{(L)} \cdot \dots \cdot \varphi(\boldsymbol{\theta}^{(1)} \cdot \boldsymbol{x})\Big),$$

form the loss $\mathcal{L}(y, y_{\text{pred}})$, and let

$$\boldsymbol{g}_t \; = \; \frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L}(y, y_{\text{pred}})$$

be the gradient w.r.t. $\boldsymbol{\theta}$. Adam then maintains two exponential moving averages,

$$\boldsymbol{m}_t = \beta_1 \, \boldsymbol{m}_{t-1} + (1 - \beta_1) \, \boldsymbol{g}_t, \qquad \boldsymbol{v}_t = \beta_2 \, \boldsymbol{v}_{t-1} + (1 - \beta_2) \, \boldsymbol{g}_t \odot \boldsymbol{g}_t,$$

with $\boldsymbol{m}_0 = \boldsymbol{v}_0 = 0$ and decay rates $\beta_1, \beta_2 \in [0, 1)$. To correct for the initial bias, one computes

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t}, \qquad \hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t}.$$

Finally, the parameter update is

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \, \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}, \tag{3.2}$$

where $\epsilon > 0$ is a small constant for numerical stability.

By combining per-parameter adaptive rates ($\hat{\boldsymbol{v}}_t$) with momentum-style smoothing ($\hat{\boldsymbol{m}}_t$) and correcting their biases, Adam often converges faster, is less sensitive to the initial choice of $\eta$, and handles sparse or noisy gradients more robustly than standard SGD.

### 3.1.5 Neural Network Techniques

Here we present some additional neural network techniques.

#### 3.1.5.1 Residual Connections

Residual connections, also known as *skip connections*, modify the standard layer-to-layer mapping by adding the input of a layer directly to its output. In a typical feedforward layer, the transformation is defined as:

$$\boldsymbol{a}^{(\ell)} = \varphi\Big(\boldsymbol{\theta}^{(\ell)} \boldsymbol{a}^{(\ell-1)}\Big),$$

whereas with a residual connection, this becomes:

$$\boldsymbol{a}^{(\ell)} = \boldsymbol{a}^{(\ell-1)} + \varphi\Big(\boldsymbol{\theta}^{(\ell)} \boldsymbol{a}^{(\ell-1)}\Big).$$

This additive shortcut allows the network to learn a residual function with respect to the input, rather than learning the full transformation directly.

#### 3.1.5.2 Convolutional Layers

A convolutional layer replaces the fully-connected affine map in forward propagation (cf. Algorithm 1) with a local filter bank. For each output position $i$ and channel $c$:

$$z_{i,c}^{(\ell)} = \sum_{k=1}^{K} \sum_{c'=1}^{C'} W_{c,c',k}^{(\ell)} \, a_{i+k-1,c'}^{(\ell-1)} + b_c^{(\ell)}, \quad a_{i,c}^{(\ell)} = \varphi\Big(z_{i,c}^{(\ell)}\Big).$$

Here $W^{(\ell)}$ are the convolutional kernels (spatial size $K$, $C'$ input channels). Convolutional layers are commonly used for image models due to the ability to capture relationships between adjacent neurons, analogous to pixels.

#### 3.1.5.3 Activation Functions

Using the generic symbol $\varphi$, common choices for activation functions are the Sigmoid; $\varphi(z) = (1+e^{-z})^{-1}$, and the ReLU; $\varphi(z) = \max(0, z)$. For the final layer in a model, one can use

- **Softmax:** For $\boldsymbol{z} \in \mathbb{R}^n$,
$$\varphi(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)},$$
so that the outputs sum to 1 across classes.
- **Gumbel-Softmax:**
$$\varphi(\boldsymbol{z})_i = \frac{\exp\Big((z_i + g_i)/\tau\Big)}{\sum_j \exp\Big((z_j + g_j)/\tau\Big)},$$
where $g_i \sim \text{Gumbel}(0, 1)$ and $\tau > 0$ is the temperature. Gumbel-Softmax is a non-determinstic activation function [17].

## 3.2 Loss functions

In this section we introduce the loss functions that will be used for the multi-class classification case.

### 3.2.1 Cross-entropy

For two distributions $p$ and $q$ defined on the same space of possible outcomes, the cross-entropy of the distribution $q$ relative to a distribution $p$ is defined as:

$$H(p, q) = -\mathbb{E}_p[\log(q)] = H(p) + D_{\text{KL}}(p||q), \tag{3.3}$$

where $\mathbb{E}_p$ is the expected value with respect to distribution $p$, $H(p)$ is the entropy of $p$ and $D_{\text{KL}}(p||q)$ is the Kullback-Leibler divergence of $p$ from $q$ [18]. It is therefore a quantification of how well the predicted distribution $q$ approximates the true distribution $p$.

In the specific case of a K-class classification task, the true distribution $p$ is represented by the one-hot encoded ground-truth labels, and the distribution $q$ by the predicted class probabilities. Therefore, the cross-entropy (CE) loss function can be defined as:

$$\mathcal{L}^{\text{CE}}(p, y) = -\sum_{i=1}^{K} p_i \log(y_i), \tag{3.4}$$

where $p_i$ is 1 if the observation belongs to class $i$ and 0 otherwise, and $y_i$ is the predicted probability for class $i$.

### 3.2.2 Max-margin loss

The max-margin (MM) loss is a logit-level alternative to cross-entropy, aiming to maximize the difference between the logit of the correct class and that of the most competitive incorrect class. In the PLG-MI paper, it is defined as:

$$\mathcal{L}^{\text{MM}}(x, c) = -l_c(x) + \max_{j \neq c} l_j(x), \tag{3.5}$$

where $l_c$ denotes the logit with respect to the target class c [4]. Moreover, the derivation of the gradient with respect to the output logits $\mathbf{o}$ is shown to be:

$$\frac{\partial \mathcal{L}^{\text{MM}}}{\partial l_k} = \begin{cases} -1 & \text{if } k = c, \\ 1 & \text{if } k = \arg\max_{j \neq c} l_j(x), \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

### 3.2.3 Vanishing Gradients

Given K classes, the one-hot encoded vector $\mathbf{y}_c = [0_1, ..., 1_c, ..., 0_K]$ of target class $c$, and the probability vector of the softmax output $\mathbf{p} = [p_1, p_2, ..., p_k]$, $p_i \in [0, 1]$ for i = 1,2,...,k, computes the partial derivative of $\mathcal{L}^{\text{CE}}$ with respect to the output logits $\mathbf{o}$ according to:

$$\frac{\partial \mathcal{L}^{\text{CE}}}{\partial \mathbf{o}} = \mathbf{p} - \mathbf{y}_c = [p_1, ..., p_c - 1, ..., p_K]. \tag{3.7}$$

It is clear that during optimization of this loss function, $p_c \to 1$ and $p_i \to 0$ for $i \neq c$ $\implies \frac{\partial}{\partial \mathbf{o}} \mathcal{L}^{\text{CE}} \to 0$. This vanishing gradient can be a limiting factor. To address this, the MM loss can be used instead, with its gradient shown in Equation 3.6. Since this gradient is piecewise constant, it can mitigate this vanishing gradient issue.

## 3.3 Gradient Boosting - XGBoost

In contrast to DNNs, Gradient Boosting algorithms are ensemble methods based on boosting weak learners, most often decision trees. Boosting works by training models sequentially, where each new model corrects the errors of the previous ones.

Among Gradient Boosting algorithms, Extreme Gradient Boosting (XGBoost) is shown by Grinsztajn et al. to be one of the best performing machine learning algorithms for classification tasks on structured tabular data. [19]

XGBoost optimizes the loss function using a Newton-Raphson-based approximation compared to standard gradient descent, allowing for faster convergence and improved computational efficiency. This second-order approximation helps refine the decision trees by considering both the gradient (first derivative) and Hessian (second derivative) of the loss function.

Because XGBoost isn't a neural network, it does not have any gradients to use for model inversion. This means that Model Inversion attacks on XGBoost target models are by definition Black-Box attacks. Therefore, it falls outside of the scope of this paper and we will not perform any attack on XGBoost models. However, since tree based models are commonly utilized, we will reference them for comparison when relevant.

### 3.3.1 Bayesian Optimization

As an alternative to gradient-based optimization like SGD and Adam, the Bayesian optimization procedure seeks a point $\mathbf{x} \in \mathcal{X}$ that minimizes a black-box objective

$$f : \mathcal{X} \to \mathbb{R}.$$

One of the methods of doing this is by using a Tree-Parzen Estimator (TPE) algorithm[20]. TPE uses two densities

$$\ell(\boldsymbol{x}) = p\Big(\boldsymbol{x} \mid f(\boldsymbol{x}) < y^*\Big), \qquad g(\boldsymbol{x}) = p\Big(\boldsymbol{x} \mid f(\boldsymbol{x}) \geq y^*\Big),$$

where $y^*$ is a performance threshold (e.g. a quantile of past losses). The next candidate point is chosen by

$$\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x} \in \boldsymbol{X}} \frac{\ell(\boldsymbol{x})}{g(\boldsymbol{x})},$$

which naturally balances sampling in regions likely to improve upon $y^\star$ against exploring elsewhere. This strategy provides an alternative to gradient-based methods when gradients are unavailable, such as for tree based models.

## 3.4 Generative Adversarial Networks

A Generative Adversarial Network is a combination of two models: a generator $G$ and a discriminator $D$. These models are trained simultaneously in a two-player game with opposing objectives. The generator $G : \mathcal{Z} \to \mathcal{X}$ is a function which maps a vector $\boldsymbol{z} \in \mathcal{Z}$, sampled from a known prior distribution (most commonly a standard normal), to a synthetic data sample $\boldsymbol{x} = G(\boldsymbol{z}) \in \mathcal{X}$, where $\mathcal{X}$ is the space of the real data [21].

In the context of GANs, the set $\mathcal{Z}$ is known as the latent space (also referred to as the embedding space), a high-dimensional vector space from which latent vectors $\boldsymbol{z} \in \mathcal{Z}$ are sampled. These latent vectors serve as inputs to the generator function $G : \mathcal{Z} \to \mathcal{X}$, mapping points from the latent space to the data space $\mathcal{X}$, thereby controlling the generation process.

The discriminator $D : \mathcal{X} \to [0, 1]$ is a function which takes a data sample $\boldsymbol{x} \in \mathcal{X}$ as input, and outputs a scalar representing the probability that $\boldsymbol{x}$ comes from the real data distribution rather than being generated by $G$. The discriminator is trained to distinguish between real samples from the dataset and fake samples from the generator. These two models act as adversaries; the generator's goal is to trick the discriminator that it has produced a real sample, and the discriminator's goal is to constantly improve it's ability to distinguish between real and generated examples.

This functionality is captured in the GAN's loss functions. The discriminator loss $\mathcal{L}_D$ consists of two terms, where the first represents its ability to identify a real sample and the second its ability to identify a generated sample. The generator loss is then determined by the discriminator output, where a lower loss represents the discriminator being fooled into believing the generated samples are real more often. Assuming $\boldsymbol{z}$ is sampled from a standard normal, the loss functions can be represented as

$$\mathcal{L}_D = -\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[\log D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(0,1)}[\log(1 - D(G(\boldsymbol{z})))] \tag{3.8}$$

$$\mathcal{L}_G = -\mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(0,1)}[\log D(G(\boldsymbol{z}))]. \tag{3.9}$$

As seen in 3.8 and 3.9, the two loss functions are defined as expectations over the data distribution and some latent space normal prior. Here, $\boldsymbol{x} \sim \mathcal{D}$ corresponds to sampling a real example from the data set, while $\boldsymbol{z} \sim \mathcal{N}(0, 1)$ denotes a standard normal latent vector. In 3.8, the term

$$-\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[\log D(\boldsymbol{x})]$$

penalizes $D$ when it assigns low real-probabilities to true data, and

$$-\mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(0,1)}[\log(1 - D(G(\boldsymbol{z})))]$$

penalizes it when it fails to reject samples $G(\boldsymbol{z})$ produced by the generator. Similarly, the generator loss in 3.9 encourages $G$ to produce outputs that the discriminator believes are real. In practice, both expectations are approximated by averaging over batches of data.

### 3.4.1 CTGAN

*CTGAN* is a conditional Generative Adversarial Network (cGAN) for tabular data introduced by Xu et al. in 2019 [22] with source code of its implementation publicly available. It is primarily used to generate synthetic data conditioned on specific values of discrete columns. Due to the purpose of generating synthetic data, CTGAN uses a Gumbel-Softmax activation function, introducing noise to the generation process.

CTGAN is a conditional generator, meaning it generates data samples $\boldsymbol{x}$ based not only on a latent vector $\boldsymbol{z}$, but also on a given condition $y$. This condition corresponds to a specific value of a categorical feature in the real data $X$ that CTGAN aims to model. Given $G(\boldsymbol{z}, y) = \boldsymbol{x}$, the generated sample $\boldsymbol{x}$ should contain the value $y$ in the feature where $y$ originally belongs.

#### 3.4.1.1 CTGAN Transformer Module

CTGAN consists of a generator and a discriminator, similarly to other GANs. In addition to these, it also trains a transformer module that represents the tabular data numerically with consideration to discrete vs. continuous features.

This module transforms structured tabular training data into an unstructured tensor format. It does this by using feature specific transformations, depending on if the feature is numerical or categorical/discrete.

Categorical/discrete features are transformed using one-hot encoding while continuous columns are modeled using a Cluster-Based Normalizer (CBN), which applies a Bayesian Gaussian Mixture Model (GMM) to partition the data into multiple Gaussian distributions. The data is then scaled to a normalized range of $[-1, 1]$, and the output is represented as a combination of the normalized value and the probability distribution over the Gaussian components using a softmax activation.

After the transformer module has been fitted in this way, the transform method $f_G$ is used to transform the training data from its raw data format to tensor format for input to the discriminator during training. The generator output can then be transformed back from tensor format to its original format with the inverse transform $f_G^{-1}$, preserving the underlying data structure.

#### 3.4.1.2 CTGAN Conditioning and Loss

During training, CTGAN randomly selects values $y$ from categorical and discrete features to use as conditions. These conditions are not enforced deterministically—no component of the architecture explicitly sets the corresponding feature in $\boldsymbol{x}$ to $y$. Instead, the generator model is guided to reproduce $y$ in the generated samples using the conditioning loss:

$$\mathcal{L}_{\text{cond}} = \mathbb{E}\left[\sum_{j \in \mathcal{S}} m_j \cdot \mathcal{L}^{\text{CE}}\left(x_j, \arg\max \boldsymbol{c}_j\right)\right] \tag{3.10}$$

where:

- $\mathcal{S}$ is the set of columns containing categorical or discrete values.
- $x_j$ is the subvector of $\boldsymbol{x}$ corresponding to column $j$,
- $\boldsymbol{c}_j$ is the one-hot encoded vector corresponding to the condition $y$, i.e., the position in $\boldsymbol{c}_j$ corresponding to the category $y$ is set to 1.
- $m_j \in \{0, 1\}$ is a mask indicating whether the condition $y$ is applied for column $j$.

Equation 3.10 therefore contributes 0 for columns where no condition is applied, and otherwise contributes the Cross-Entropy loss between the target value and the generated value $x_j$.

## 3.5 Semantics of Tabular Data

In this section, we define the semantics of the tabular data representation adopted in our work. This is in order to avoid confusion between our definitions and other works that might have broader or narrower definitions of these terms. Tabular data is organized into rows and columns, where each column represents a feature and each row corresponds to a sample or instance.

- **Features:** Each column in the table is associated with a random variable $X$, interchangeably referred to as a feature. A feature captures a particular aspect or attribute of the data across all samples.
- **Samples:** Each row corresponds to a single sample (or data point/observation). That is, a row represents one outcome across all features.

We distinguish between different types of features based on the nature of their values:

- **Numerical features** are features with values that are inherently quantitative and carry numerical meaning. These are divided into:
  - **Continuous features** are features whose values are continuous, $X \in \mathbb{R}$.
  - **Discrete features** are features whose values are integers, $X \in \mathbb{Z}$. They often represent counts or other non-continuous numerical quantities. It is possible to measure a difference between discrete features.
- **Categorical features** are features whose values are elements from a finite set of categories or labels. The values themselves do not carry inherent numerical meaning, and are often nominal or ordinal in nature. It is not possible to measure a difference between categorical features.
- **Boolean features** are a subset of categorical features, where there are only two categories, true or false.

It is important to emphasize that, within the scope of our definitions, *categorical features are not equivalent to discrete features*. Although both may take on a finite set of values, categorical features represent labels or classes without an inherent order or structure, whereas discrete features are numerical and typically ordered.

## 3.6 Evaluation Metrics

Here we present the evaluation metrics used for reconstruction attacks in both image and tabular data applications.

### 3.6.1 Attack Accuracy

In the context of reconstruction attacks, attack accuracy refers to the ability of a model to correctly identify the intended class of a reconstructed sample. This is evaluated using an evaluation model $E$ with the same classification task and training set as the target model $T$. Because reconstruction attacks use properties of the target model, the evaluation model is usually of a different architecture in order to reduce biases and dependencies of the target model. Attack accuracy evaluated on $N$ reconstructed samples can be expressed as:

$$\frac{1}{N}\Sigma_{i=0}^{N}\,\mathbf{I}(E(\boldsymbol{x}_i) = y_i) \quad \text{where} \quad \boldsymbol{x}_i = G(\boldsymbol{z}_i, y_i). \tag{3.11}$$

In this setting, the attack accuracy measures how often reconstructed samples $\boldsymbol{x}_i$ conditioned on $y_i$ are classified by the evaluation model $E$ as belonging to their intended label $y_i$. A high evaluation accuracy indicates that the reconstructed samples retain sufficient class-discriminative features to be recognized as the correct class by $E$.

### 3.6.2 Frechét Inception Distance

The Frechét Inception Distance (FID) is a metric to evaluate how much the distributions of two sets of images differ, introduced by Heusel et al. in 2017 [23]. This is done by utilizing the final layer of an Inception-v3 [24] model to find a feature space for both distributions. The feature spaces are then approximated by two multivariate Gaussians $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\mathcal{N}(\boldsymbol{\mu}', \boldsymbol{\Sigma}')$ from which the Frechét Distance can be calculated as follows:

$$d_F(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \mathcal{N}(\boldsymbol{\mu}', \boldsymbol{\Sigma}'))^2 = \|\boldsymbol{\mu} - \boldsymbol{\mu}'\|_2^2 + \operatorname{tr}\left(\boldsymbol{\Sigma} + \boldsymbol{\Sigma}' - 2\left(\boldsymbol{\Sigma}\boldsymbol{\Sigma}'\right)^{\frac{1}{2}}\right) \tag{3.12}$$

The FID metric is a distance metric between the reconstructed and private data, where a lower distance indicates a higher similarity between the two and therefore a more accurate reconstruction.

### 3.6.3 k-NN Distance

The k-NN distance metric for image reconstruction calculates the average distance between the reconstructed image and its k nearest neighbors in the training set:

$$\mathrm{d}_{\text{k-NN}}(S, \tilde{\boldsymbol{x}}, K) = \frac{1}{K}\sum_{s \in S} \operatorname{Dist}(s, \tilde{\boldsymbol{x}}), \tag{3.13}$$

where $\tilde{\boldsymbol{x}}$ the reconstructed data and S is the subset of the training (private) data which includes the K closest data to $\tilde{\boldsymbol{x}}$. The distance function commonly used is the $\ell_2$ distance.

For two images with N pixels, the $\ell_2$ distance measures the pixel-wise differences between two images, quantifying the overall error or distortion:

$$\ell_2 = \sqrt{\sum_{i=1}^{N}((I_1(\mathrm{i}) - I_2(\mathrm{i}))^2}, \tag{3.14}$$

where $I_1$ and $I_2$ are the pixel intensity vectors of the first and second image, respectively. Similarly to FID, KNN distance is a distance metric between the reconstructed and private data, where a lower number corresponds to a closer reconstruction.

### 3.6.4 Column Shape Scores

To evaluate the success of a model inversion reconstruction attack for tabular data, the statistical similarity of the real (private) and synthetic (reconstructed) data can be compared. One such comparison is to analyze the shape similarity of the marginal distributions of individual columns. Therefore, for each column, a real and synthetic distribution $\pi_R(\boldsymbol{x})$ and $\pi_S(\boldsymbol{x})$, with $\boldsymbol{x}$ being a realization of $X$, can be considered and evaluated using an appropriate metric, depending on whether the independent and identically distributed (i.i.d) random variable $X$ is continuous or discrete.

These metrics are readily available and intuitively accessible in the Python library *The Synthetic Data Vault* (SDV) by DataCebo [25].

#### 3.6.4.1 Kolmogorov-Smirnov Statistic

The Kolmogorov-Smirnov statistic (KSS) is a metric that is applicable to numerical columns. Let $F_{R_n}(\boldsymbol{x})$ and $F_{S_m}(\boldsymbol{x})$ denote the empirical cumulative distribution functions (CDFs) of the probability density functions $R(\boldsymbol{x})$ and $S(\boldsymbol{x})$ for the real and synthetic column distributions for an i.i.d random real variable $X$, with $n$ and $m$ observations, respectively. The two-sample KSS is defined as:

$$D_{n,m} = \sup_{\boldsymbol{x}} \left| F_{R_n}(\boldsymbol{x}) - F_{S_m}(\boldsymbol{x}) \right| . \tag{3.15}$$

Hence, KSS also captures higher moments of the distributions, including skewness (third standardised moment) and kurtosis (fourth standardised moment). Therefore, it also provides a measure of shape similarity between the distributions $\pi_R(\boldsymbol{x})$ and $\pi_S(\boldsymbol{x})$.

This statistic can be converted into a shape similarity score, called *KSComplement* in SDV, in the interval $[0, 1]$ by evaluating the complement $1 - D_{n,m}$, where a value of 1 indicates maximum similarity between the two distributions.

#### 3.6.4.2 Total Variation Distance

The total variation distance is a metric that is applicable to categorical and boolean columns. Let $K$ be an i.i.d discrete random variable, and let $R(k)$ and $S(k)$ denote the real and synthetic probability mass functions of the column distributions, respectively. These are defined over a sample space ($\Omega$), where $\Omega = \{0, 1\}$ for boolean columns and $\Omega = \{0, 1, ..., C - 1\}$ for categorical columns with C distinct categories. The total variation distance between $R(k)$ and $S(k)$ is defined as:

$$\delta(R, S) = \frac{1}{2} \sum_{\omega \in \Omega} |R(\omega) - S(\omega)|, \tag{3.16}$$

where $\omega \in \Omega$ represents a specific outcome in the sample space for the random variable $K$, and $R(\omega)$ and $S(\omega)$ the corresponding probabilities for that outcome. This distance can be converted into a shape similarity score, called *TVComplement* in SDV, in the interval $[0, 1]$ by evaluating the complement $1 - \delta(R, S)$, where a value of 1 indicates maximum similarity between the two distributions $R(k)$ and $S(k)$.

### 3.6.5 Column Pair Trends

In addition to evaluating the marginal distributions of individual columns, the pairwise correlations of the columns can also be assessed by comparing reconstructed and private data. This evaluation helps in determining whether or not the attack successfully has reconstructed joint relationships that exist in the private data. Again, depending on the data types of the column pairs, different similarity measures can be used to quantify this pairwise alignment.

### 3.6.6 Pearson's Correlation Coefficient

Given a pair of two i.i.d random variables X and Y, Pearson's correlation coefficient $\rho_{X,Y}$, which captures linear trends between pairs of random variables, is given by:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}, \tag{3.17}$$

where $\mu_X$ and $\mu_Y$ is the mean of X and Y, respectively, and $\sigma_X$, $\sigma_Y$ are the corresponding standard deviations.

### 3.6.7 Correlation Similarity

Correlation similarity is a metric applicable to numerical column pairs. Consider a pair of two i.i.d random real variables X and Y. Let $\rho^{(S)}_{X,Y}$ and $\rho^{(R)}_{X,Y}$ denote Pearson's correlation coefficient for the synthetic and real data, respectively. The absolute deviation $\Delta(S,R)$ is:

$$\Delta(S,R) = \frac{\mid \rho^{(S)}_{X,Y} - \rho^{(R)}_{X,Y} \mid}{2}. \tag{3.18}$$

Note that $\Delta(S,R)$ is normalized to the interval $[0,1]$. To get a similarity score, compute $1 - \Delta(R,S)$, where a value of 1 indicates maximum similarity in linear correlation between the variables for the synthetic and real samples.

### 3.6.8 Contingency Similarity

Contingency similarity is a metric applicable to categorical and boolean columns. Consider a pair of two i.i.d discrete random variables X and Y for a pair of columns in the synthetic and real samples. Let $R(x,y)$ and $S(x,y)$ denote the joint probability mass functions of the real and synthetic column distributions, respectively. Using similar notation as in Section 3.6.4.2, the total variation distance between the real and synthetic joint distributions can then be computed by:

$$\delta(R,S) = \frac{1}{2} \sum_{\omega_X \in \Omega_X} \sum_{\omega_Y \in \Omega_Y} \mid R(\omega_X,\omega_Y) - S(\omega_X,\omega_Y) \mid . \tag{3.19}$$

This distance can be converted into a similarity score in the interval $[0,1]$ by evaluating $1 - \delta(R,S)$, where a value of 1 indicates maximum similarity in correlation between the variables for the synthetic and real samples.

### 3.6.9 Gower's Distance

Gower's distance is a flexible dissimilarity measure for tabular data with mixed variable types. It combines per-feature distances between two observations into an overall score, taking values in $[0, 1]$ (where 0 means the 2 points are identical and 1 means maximally different). Gower's distance can also be used as a similarity measure, where 1 would indicate identical points and 0 maximally different, however we choose to implement the dissimilarity measure for our experiments. We use the average definition, treating all features with equal weight in the sum.

Let observations $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$ each have $p$ features. For feature $k$ we compute a partial distance $d_{ijk} \in [0, 1]$, then define the overall Gower's distance as

$$D_{ij} \;=\; \frac{1}{p} \sum_{k=1}^{p} d_{ijk}. \tag{3.20}$$

Each partial distance $d_{ijk}$ depends on the type of feature $k$:

If feature $k$ is numeric,

$$d_{ijk} = \frac{|\, x_k^{(i)} - x_k^{(j)}|}{R_k}. \tag{3.21}$$

where $R_k = \max(x_k) - \min(x_k)$ is the observed range of feature $k$, ensuring that $d_{ijk} \in [0, 1]$.

If feature $k$ is discrete, then :

$$d_{ijk} = \begin{cases} 0, & x_k^{(i)} = x_k^{(j)}, \\ 1, & x_k^{(i)} \neq x_k^{(j)}. \end{cases} \tag{3.22}$$

Equation (3.20) then yields a single dissimilarity score $D_{ij}$ between two observations $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$.

# 4

# Methodology

In this section we introduce how the PLG-MI attack is performed on both image and tabular data, and the corresponding experimental setup.

## 4.1 Experimental setup

Our experiments were run on the following system:

- **Operating system:** Ubuntu 24.04 LTS
- **GPU:** NVIDIA Tesla T4, 16 GB
- **CUDA driver:** CUDA 12.8 (Driver 570.86.15)
- **Python:** version 3.12.8

## 4.2 PLG-MI Attack

The Pseudo-label Guided Model Inversion Attack (PLG-MI) was introduced by Yuan et al. in 2023 [4]. This attack is shown to be one of the strongest state-of-the-art image reconstruction attacks by Qiu et al. [14], especially in the lower resolution setting.

The PLG-MI attack uses a conditional GAN (cGAN) that is trained on a public data set. This conditioning is done by pseudo-labeling the public data set. The pseudo-labels are assigned by querying the target model with the full public data set, and assigning each data point to the class of the highest target model confidence. The pseudo-labeling technique is utilized in many subsequent works [13],[26].

In PLG-MI, the pseudo-labeling is then refined by a top-$n$ selection, choosing the top-$n$ highest assigned confidence data points of each class to use as training data for the GAN and dismissing the low-confidence examples. This step accomplishes two things; it removes the influence of low-confidence pseudo-labels that could be misleading when training the GAN, and it reduces potential class imbalance in the data set used for training the GAN.

Conditioning the training of the GAN on these pseudo-labels enables it to learn more specific and independent data distributions for each class. This class information is embedded into the latent space, which can force generated samples towards specific classes in the private data. In essence, the GAN latent space is divided into separate

**Algorithm 2** PLG-MI Training Phase: Conditional GAN Training

**Input:** Generator $G$, Discriminator $D$, Target model $T$, Pseudo-labeled public data $\mathcal{D}_{\text{pseudo}}$, Generator loss $\mathcal{L}_G$, Discriminator loss $\mathcal{L}_D$, Inversion loss $\mathcal{L}_{\text{inv}}$, Stochastic Augmentations $f_{\text{aug}}$, Mixing weight $\alpha$, Iterations $N$, Discriminator steps per iteration $n_D$

```
for i = 0 to N do
    for j = 0 to n_D do
        if j = 0 then
            Sample fake images and labels: (x_fake, y_fake) ~ G
            Apply augmentations: x̃_fake = f_aug(x_fake)
            Compute discriminator output: D_fake = D(x_fake)
            Compute inversion loss: L_inv = L_inv(T(x̃_fake), y_fake)
            Compute generator loss: L_G = L_G(D_fake)
            Combine losses: L_all = L_G + α L_inv
            Compute gradient:
```

$$\nabla_{\boldsymbol{\theta}_G}\mathcal{L}_{\text{all}} = \frac{\partial}{\partial \boldsymbol{\theta}_G}\Big(-D(G(\boldsymbol{z}, y)) + \alpha \cdot \mathcal{L}^{\text{MM}}(T(G(\boldsymbol{z}, y)))\Big)$$

```
            Update generator parameters: θ_G ← θ_G − η_G ∇_{θ_G} L_all
        end if
        Sample fake batch: (x_fake, y_fake) ~ G
        Sample real pseudo-labeled batch: (x_real, y_real) ~ D_pseudo
        Compute: D_fake = D(x_fake), D_real = D(x_real)
        Compute discriminator loss: L_D = L_D(D_fake, D_real)
        Update discriminator: θ_D ← θ_D − η_D ∇_{θ_D} L_D
    end for
end for
```

**Output:** Trained Generator $G$ and Discriminator $D$

class subspaces, which confines the search of the latent space to those subspaces when learning features of each class.

For the generator loss function, one uses the Max-Margin loss function from Equation 3.5, and the expectation of the discriminator output from Equation 4.1. These are then added together using a scaling parameter $\alpha$, see Equation 4.2.

$$\mathcal{L}_{\text{inv}} = \mathcal{L}^{\text{MM}}(T(G(\boldsymbol{z}, y)), y), \quad \mathcal{L}_G = -\mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(0,1)}\left[D(G(\boldsymbol{z}, y))\right] \tag{4.1}$$

$$\mathcal{L}_{\text{all}} = \mathcal{L}_G + \alpha\mathcal{L}_{\text{inv}} \tag{4.2}$$

The loss function $\mathcal{L}_{\text{all}}$ (Eq. 4.2) is used to incorporate the predictions of the target model into the loss of the generator used for backpropagation. This is to incentivize the generator to generate samples that are accurate to the target model. The GAN

**Algorithm 3** PLG-MI Attack Phase: Latent Space ($\boldsymbol{z}$) Optimization

**Input:** Target model $T$, Trained Generator $G$, Learning rate $\eta$, Stochastic Augmentations $f_{\text{aug}}$, Number of iterations $N$
Initialize latent vector: $\boldsymbol{z} \sim \mathcal{N}(0,1)^d$, corresponding target label $y$.

**for** $i = 0$ to $N$ **do**
  Generate fake images: $\boldsymbol{x} = G(\boldsymbol{z}, y)$
  Compute model outputs:
  $y_{\text{pred}}^{(1)} = T(f_{\text{aug}}(\boldsymbol{x})), \quad y_{\text{pred}}^{(2)} = T(f_{\text{aug}}(\boldsymbol{x}))$
  Compute inversion loss:
  $\mathcal{L}_{\text{inv}} = \mathcal{L}^{\text{CE}}(y_{\text{pred}}^{(1)}, y) + \mathcal{L}^{\text{CE}}(y_{\text{pred}}^{(2)}, y)$
  Compute gradient:

$$\nabla_{\boldsymbol{z}} \mathcal{L}_{\text{inv}} = \frac{\partial \mathcal{L}_{\text{inv}}}{\partial y_{\text{pred}}} \cdot \frac{\partial y_{\text{pred}}}{\partial \boldsymbol{x}} \cdot \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{z}}$$

  (via backprop. through $T$ and $G$)
  Update latent vector $\boldsymbol{z}$:
  $\boldsymbol{z} \leftarrow \boldsymbol{z} - \eta \nabla_{\boldsymbol{z}} \mathcal{L}_{\text{inv}}$
**end for**

**Output:** Optimized latent vector $\boldsymbol{z}^*$

is then trained according to Algorithm 2. In the original implementation, stochastic augmentations $f_{\text{aug}}$ are applied to the generated samples used as input to the target model, to avoid overfitting. In the image case, these are combinations of small random crops, rotations and color shifts. This is averaged over 2 different samples, with different outcome of $f_{\text{aug}}$, the stochastic augmentations.

The attack phase of the PLG-MI attack consists of an optimization of the latent space vectors $\boldsymbol{z}$ to find the highest probability vectors $\boldsymbol{z}^*$ for each class. These optimized $\boldsymbol{z}$ are meant to represent the most class-typical points in the latent space for each generated sample. When these are found, the final reconstructed images can be generated. The attack iteratively updates $\boldsymbol{z}$ using an *Adam* optimizer, and based on a Cross-entropy loss of the generated output, according to Algorithm 3. Due to the stochastic augmentations $f_{\text{aug}}$, the loss is averaged over two samples. When $\boldsymbol{z}^*$ is found, the PLG-MI attack is complete and the reconstructed images are generated by $\boldsymbol{x} = G(\boldsymbol{z}^*, y)$. For a flow-chart description of the PLG-MI attack pipeline, we recommend Figure 3 in the PLG-MI paper[4].

## 4.3 Adapting PLG-MI to Tabular Data

In the PLG-MI paper, the application of image reconstruction is implemented and discussed. However, there are only two assumptions that are image domain specific. The first assumption is the specific cGAN used. In order to reconstruct tabular data, we therefore propose replacing this with a conditional tabular GAN (CTGAN) [22]. Similarly to cGAN, CTGAN can be conditioned on discrete values. By using the pseudo-labels from PLG-MI as conditioning for the CTGAN, this same functionality can be achieved. The second assumption is in the latent vector optimization step, where a gradient descent algorithm is used based on the output of the target model. This requires some adaptations for the data formats used, to allow for computational graphs to be built.

To train CTGAN according to the PLG-MI attack, we made two key adaptations:

### 4.3.1 Conditioning CTGAN for PLG-MI

The conditioning strategy during training was modified to rely solely on pseudo-labels. Typically, CTGAN conditions on all dataset features at random to learn how to generate samples conditioned on any combination of features. However, since our sampling process only conditions on pseudo-labels, we enforced the same constraint during training.

This was done by creating custom condition vectors used as input for the GAN along with the latent space input $\boldsymbol{z}$. In addition, we needed to create a unique mask for the CTGAN conditioning loss in equation 3.10 to function as intended. This included accessing properties of the CTGAN input transformations $f_G$.

### 4.3.2 CTGAN loss functions for PLG-MI

We incorporate the inversion loss function from Equation 4.1 for the target model to guide the training process. In addition to the generator loss and inversion loss, the conditioning loss from equation 3.10 is used. Combined, this means that we use 3 terms in the loss function for our CTGAN,

$$\mathcal{L}_{\text{all}} = \mathcal{L}_G + \alpha_1 \mathcal{L}_{\text{inv}} + \alpha_2 \mathcal{L}_{cond}. \tag{4.3}$$

Each term plays a distinct role in the training process. The generator loss ($\mathcal{L}_g$) encourages the generator to produce realistic samples that fool the discriminator. The inversion loss ($\mathcal{L}_{\text{inv}}$) ensures that the output of the target model aligns with the conditioned labels. Finally, the conditioning loss ($\mathcal{L}_{\text{cond}}$) drives the generated samples themselves to match the specified conditioned labels.

### 4.3.3 Adaptations of tabular models for gradient propagation

In the original adaptation of both the CTGAN model as well as the PyTorch-Tabular model, each includes functionality to convert between PyTorch tensor format and pandas DataFrame format. When using each model on their own, these transformations are necessary because the data set is in the structured tabular pandas DataFrame format, and the underlying neural network models operate on unstructured tensor data. However, in our attack these transformations prohibit back-propagation, due to pandas DataFrames not being able to support a computational graph.

**Algorithm 4** Naive PLG-MI Tabular Latent Space ($\boldsymbol{z}$) Optimization (with DataFrame conversion)

**Input:** Target model $T$, trained generator $G$, learning rate $\eta$, augmentations $f_{\text{aug}}$, CTGAN tensor→DataFrame (inverse) transformation $f_G^{-1}$, Target Model DataFrame→tensor transformation $f_T$, target label $y$.

**Initialize:** $\boldsymbol{z} \sim \mathcal{N}(0,1)^d$.

Generate activations: $\boldsymbol{h} = G(\boldsymbol{z}, y)$

Convert to DataFrame sample: $\boldsymbol{x} = f_G^{-1}(\boldsymbol{h})$

Convert back to tensor and input to target model: $y_{\text{pred}} = T(f_T(\boldsymbol{x}))$

Compute inversion loss: $\mathcal{L}_{\text{inv}} = \mathcal{L}^{\text{CE}}(y_{\text{pred}}, y)$

Compute gradient via backprop:

$$\nabla_{\boldsymbol{z}} \mathcal{L}_{\text{inv}} = \frac{\partial \mathcal{L}_{\text{inv}}}{\partial y_{\text{pred}}} \frac{\partial y_{\text{pred}}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}$$

(Note: $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{h}}$ and $\frac{\partial y_{\text{pred}}}{\partial \boldsymbol{x}}$ are blocked by the tensor/pandas conversion.)

Update latent: $\boldsymbol{z} \leftarrow \boldsymbol{z} - \eta \nabla_{\boldsymbol{z}} \mathcal{L}_{\text{inv}}$

**Output:** Optimized latent $\boldsymbol{z}$

In our latent space optimization step, these transformations between data formats are redundant. As shown in Algorithm 4, the naive approach uses two format conversion steps $f_G^{-1}$ and $f_T$ which block the flow of the gradient. Here we denote the tensor activation before transformation with $\boldsymbol{h}$. The redundancy arises because $f_G^{-1}$ transforms a tensor into a DataFrame, and $f_T$ then transforms this back to tensor format, without performing any other operations on the intermediate DataFrame. Our solution to this problem, as seen in Algorithm 5, replaces both with a single differentiable inverse transform $f_{\text{inv}}$, restoring an unbroken computational graph from $\boldsymbol{z}$ to the final loss. The challenge with this approach is to exactly replicate $f_T(f_G^{-1}(\boldsymbol{h}))$ with $f_{\text{inv}}(\boldsymbol{h})$ while keeping all transformations involved with tensor operations. These transformations are normalizations and encodings.

In addition to the technical challenges of constructing $f_{\text{inv}}$, this process also assumes that knowledge about the target model transformation from DataFrame to tensor, $f_T$, is available to the adversary.

For our optimization method to function, the CTGAN Gumbel-Softmax activation function was replaced with a regular Softmax function. This was necessary in order to have a deterministic generation process, so that for a fixed $\boldsymbol{z}$ and $y$, the generation $G(\boldsymbol{z}, y) = \boldsymbol{x}$ produces the same $\boldsymbol{x}$ each time. The noise introduced by the Gumbel-Softmax activation function made it difficult to reliably optimize $\boldsymbol{z}$, since changes in the output $\boldsymbol{x}$ could not be attributed solely to changes in $\boldsymbol{z}$.

### 4.3.4 Alternative latent space optimization methods

Since the original latent space optimization method becomes more complex for tabular data, as described above, we also propose 2 alternative techniques.

The first technique is to skip the optimization step altogether and use randomized

**Algorithm 5** Pure-Torch PLG-MI Tabular Latent Space ($\boldsymbol{z}$) Optimization (with differentiable $f_{\text{inv}}$)

**Input:** Target model $T$, trained generator $G$, learning rate $\eta$, augmentations $f_{\text{aug}}$, differentiable inverse-transform $f_{\text{inv}} \sim f_T \circ f_G^{-1}$, target label $y$.
**Initialize:** $\boldsymbol{z} \sim \mathcal{N}(0,1)^d$.

1: Generate activations: $\boldsymbol{h} = G(\boldsymbol{z}, y)$
2: Transform directly to $T$-compatible tensor and predict: $y_{\text{pred}} = T(f_{\text{inv}}(\boldsymbol{h}))$
3: Compute inversion loss: $\mathcal{L}_{\text{inv}} = \mathcal{L}^{\text{CE}}(y_{\text{pred}}, y)$
4: Compute gradient via backprop:

$$\nabla_{\boldsymbol{z}} \mathcal{L}_{\text{inv}} = \frac{\partial \mathcal{L}_{\text{inv}}}{\partial y_{\text{pred}}} \frac{\partial y_{\text{pred}}}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}$$

(Note: $\frac{\partial y_{\text{pred}}}{\partial \boldsymbol{h}}$ is now possible through $f_{\text{inv}}$.)
5: Update latent code: $\boldsymbol{z} \leftarrow \boldsymbol{z} - \eta \nabla_{\boldsymbol{z}} \mathcal{L}_{\text{inv}}$

**Output:** Optimized latent $\boldsymbol{z}$

$\boldsymbol{z}$. This approach is also a useful sanity check to evaluate if the other optimization step is able to improve the results. The second technique is to optimize $\boldsymbol{z}$ using the bayesian approach by using the *optuna* library, that utilizes the Tree-structured Parzen Estimator algorithm. Here, we simply use the cross-entropy inversion loss $\mathcal{L}_{\text{inv}}$ as our objective and $\boldsymbol{z}$ as the parameter.

This approach allowed us to still have the same loss function as the objective to minimize, but with a different optimization algorithm. Because this Bayesian approach does not utilize the target model gradients, it can theoretically also be used with an XGBoost classifier.

## 4.4 Datasets

In this section, we present the datasets and associated methodology used in our experiments. Grouping our experiments by the dataset used serves several purposes: it represents a specific target model objective, it reflects differences in data modality, and it determines model architecture and hyperparameter selection. By structuring our evaluation in this way, we ensure that comparisons are made within the same domain and task.

### 4.4.1 CelebA

The CelebA dataset [27] consists of 202,599 face images of 10,177 unique identities with coarse alignment. The images are $178 \times 218$ color images. The images of an identity are taken in a wide variety of settings, but they portray the same celebrity.

CelebA is widely used as the standard dataset in model inversion attacks due to the higher number of classes compared to other similar data sets. Since the original PLG-MI paper reported results on CelebA[4], replicating their setup allowed us to directly

**Table 4.1:** GAN training hyperparameters and their values.

| Symbol | Value | Description |
|---|---|---|
| `top_n` | 30 | Top-$n$ samples per class |
| $n_{\text{dis}}$ | 5 | Discriminator updates per generator update |
| $n_{\text{iter}}$ | 30,000 | Total training iterations |
| `batch_size` | 64 | Number of samples per training batch |
| $\eta_{\text{gen}}$ | 0.0002 | Generator learning rate |
| $\eta_{\text{dis}}$ | 0.0002 | Discriminator learning rate |
| $\beta_1$ | 0.0 | Adam optimizer $\beta_1$ parameter (shared) |
| $\beta_2$ | 0.9 | Adam optimizer $\beta_2$ parameter (shared) |
| $\text{dim}_z$ | 128 | Dimension of latent vector $\boldsymbol{z}$ |
| $\alpha$ | 0.2 | Inversion loss scaling parameter |

compare our outputs with those published in the original work. This verification step ensured that our implementation aligns with previous results before extending the attack to other domains and datasets.

The target model task for the CelebA data set is to predict the identity of an input image. The input images are downsampled to a resolution of $64 \times 64$.

The dataset was split so that 1,000 identities were assigned as the private dataset $\mathcal{D}_{\text{priv}}$ and the complementary set was assigned as the public data set $\mathcal{D}_{pub}$, in accordance with the CelebA setup in the PLG-MI paper. In contrast to the paper, which uses a 30,028 image 1,000 identity subset as the private data set, we use identities labeled 1 to 1000 for $\mathcal{D}_{\text{priv}}$. This selection method yielded a private data set of 21,152 images in total. We made this adjustment to enhance reproducibility, as the authors of the PLG-MI paper published their private dataset but did not provide the mapping of their labels to the original CelebA dataset, making direct replication challenging. By selecting identities 1–1000 we end up with some identities contributing far more images than others—intentionally creating an imbalanced dataset. We adopt this strategy not only to investigate how class imbalance affects the PLG-MI attack, but also to align with our investigation on tabular data (MIMIC-IV), which is likewise unbalanced.

A ResNet-18 and a ResNet-152 were used as target models, trained on a stratified 90% split of the private data set with the remaining 10% as validation. Both of these were initialized on PyTorch ImageNet1K weights prior to training on $\mathcal{D}_{\text{priv}}$. For evaluation, a ResNet-50 model was employed for both target models. This approach is a simpler implementation compared to the FaceNet IR-50 evaluation model used by the PLG-MI paper. We opted for this choice to simplify implementation within the PyTorch framework, facilitating direct comparisons with the target models. However, this design choice also increases architectural alignment between the target and evaluation models, potentially introducing statistical dependencies between them.

In accordance with the PLG-MI paper, we used the parameters listed in table 4.1.

### 4.4.2 MIMIC-IV

Multi-parameter Intelligent Monitoring for Intensive Care-IV (MIMIC) is the 4th edition of the MIMIC tabular data set by Johnson et al. [28]. This data set contains multiple tables of de-identified hospital records collected from Beth Israel Deaconess Medical Center.

Due to the size and number of tables in MIMIC-IV, some careful curation and selection needed to be done in order to suit our task.

There are many possible classification tasks one could perform on the MIMIC-IV data set. We chose the diagnosis code given during each hospital admission as our target label for classification. This task was motivated by the large number of possible diagnoses, and that it has resemblance to real-world diagnosis prediction classifiers.

Both a Multi-Layer-Perceptron and a GANDALF model (see 4.5.3) were used as target models on this dataset, with the other being used as evaluation model. Both models trained on a stratified train-test split, with 95% of the private data allocated to training and 5% to testing. 20% of the training data was reserved for validation. The MLP model consisted of a feed-forward neural network with four hidden layers of sizes 2,048, 1024, 512 and 256, using ReLU activations. The GANDALF model was configured with 16 GFLU stages, a GFLU dropout rate of 0.1 and an embedding dropout of 0.1. All models were trained using the PyTorch Tabular framework [29] with a batch size of 256 and a set maximum of 150 training epochs. The early-stopping metric was set to *train_loss_0*, which is the output of the first training loss output head. No improvement of 0.1% over the last 3 epochs of this metric will lead to early-stopping. Continuous features were not normalized prior to training. All other parameters were set to default in the PyTorch Tabular Framework (version 1.1.1).

#### 4.4.2.1 Data preparation

For an overview of the full data processing pipeline, see Figure A.1.

**Number of Medications:** This engineered feature is derived from a table that records which medications were administered to each patient, along with their quantities. Each row in the table corresponds to one administration event. For this feature, the temporal aspect and dosage were disregarded; instead, the number of unique medications administered to a patient was counted. In addition, the specific medications were retained as individual features. To limit the number of features, only the 100 most frequently occurring medications were included.

**Number of Procedures:** This feature was constructed using a table containing laboratory events. The total number of procedures recorded for each patient was extracted. Furthermore, the individual procedures were encoded as binary features, but only events where the 'flag' attribute was marked as *abnormal* were considered. Multiple rows corresponding to the same patient were aggregated; for binary features, a value of *True* was retained if it appeared in any of the patient's rows.

**General Preprocessing:** Extreme outliers were removed. Missing values in nu-

**Table 4.2:** Feature overview with data types using the described feature selection. The shape of the resulting total data set is 592,131 rows × 587 columns. This total data set is then further split into public and private data sets according to Table 4.3.

| Feature | Data type |
|---|---|
| `Weight (Lbs)` | float64 |
| `BMI (kg/m`$^2$`)` | float64 |
| `Height (Inches)` | float64 |
| `eGFR` | float64 |
| `systolic` | float64 |
| `diastolic` | float64 |
| `num_medications` | int64 |
| `num_procedures` | int64 |
| `length_of_stay` | int64 |
| `race` | category |
| `insurance` | category |
| `gender` | category |
| `curr_service_`$\mathtt{i}_0$`-`$\mathtt{i}_{21}$ | bool |
| `med_`$\mathtt{j}_0$`-`$\mathtt{j}_{99}$ | bool |
| `lab_event_`$\mathtt{k}_0$`-`$\mathtt{k}_{451}$ | bool |
| `diagnosis code (target)` | int64 |

merical columns were imputed using either the column mean or by modeling the column as a normal distribution and sampling from it. Categorical features were transformed using one-hot encoding. At this stage, the dataset contained no missing values.

**Feature Selection:** The resulting dataset contained 587 features. To identify a subset of potentially informative features, a random forest estimator was trained on the full dataset, and feature importance scores were extracted using the mean decrease in impurity, as discussed in [30]. A subset of features was then selected with the aim of minimizing the number of features while preserving the model's predictive performance.

For an overview of the dataset after the data preparation steps, see Table 4.2. Note that in this table, the target column used for classification is the diagnosis code. Accordingly, references to "classes" in the context of this dataset pertain to the distinct diagnosis codes.

Various data preparation configurations were tested by varying the number of features selected to be included in the private/public datasets, and the number of classes included in the private dataset, to evaluate their impact on the attack. Each setting is assigned an ID that is referred to when presenting results. When selecting k classes, we picked the top-k classes, that is the classes with the most entries per class when selecting $\mathcal{D}_{\text{priv}}$, and then assigning the complimentary set as $\mathcal{D}_{\text{pub}}$. The different settings explored are summarized in table 4.3. If the target model chosen

**Table 4.3:** Overview of Data Preparation Settings.

| ID | Description | Private Data Rows |
|---|---|---|
| 0 | 100 classes, 50 features, target model: MLP | 184 471 |
| 1 | 50 classes, 50 features, target model: MLP | 136 771 |
| 2 | 20 classes, 50 features, target model: MLP | 82 922 |
| 3 | 20 classes, 100 features, target model: MLP | 82 922 |
| 4 | 20 classes, 50 features, target model: GANDALF | 82 922 |

**Table 4.4:** CTGAN training hyperparameters to be explored.

| Symbol | Description |
|---|---|
| `top_n` | Top-$n$ samples per class |
| $n_{\text{dis}}$ | Discriminator updates per generator update |
| $n_{\text{iter}}$ | Total training iterations |
| `batch_size` | Number of samples per training batch |
| $\eta_{\text{gen}}$ | Generator learning rate |
| $\lambda_{\text{gen}}$ | Generator learning rate decay |
| $\eta_{\text{dis}}$ | Discriminator learning rate |
| $\lambda_{\text{dis}}$ | Discriminator learning rate decay |
| $\beta_1$ | Adam optimizer $\beta_1$ parameter (shared) |
| $\beta_2$ | Adam optimizer $\beta_2$ parameter (shared) |
| $\text{dim}_z$ | Dimension of latent vector $z$ |
| $\alpha_1$ | Inversion loss scaling parameter |
| $\alpha_2$ | Conditioning loss scaling parameter |

is an MLP, the evaluation model is GANDALF, and vice versa.

#### 4.4.2.2 Hyperparameter Fine-tuning

The CTGAN training hyperparameters to be explored are listed in Table 4.4. Due to the lack of previous works for this particular data set and task, hyperparameter fine-tuning was warranted to ensure proper and converging training dynamics. This was done by trial and error. Any hyperparameters not shown are set to the default values provided by CTGAN version 0.11.0.

## 4.5 Target Models

In this section we present the target models used. All target models were trained to maximize training accuracy to investigate the possible effects of over-fitting on the attacks.

### 4.5.1 ResNet

For images (CelebA), we utilize the ResNet family of target models [31] as implemented in the PyTorch library. ResNets are built from blocks that include residual connections: each block passes its input forward unchanged alongside the output of a small convolutional subnetwork. This identity shortcut allows gradients to propagate more directly through the network, mitigating vanishing-gradient issues in deep architectures. We use the ResNet-18, 50 and 152 variants where the name corresponds to the number of blocks, a higher number corresponds to a deeper network.

### 4.5.2 Multi-Layer Perceptron

For tabular data (MIMIC), we utilize a Multi-Layer Perceptron (MLP). MLP is a simple feed-forward network without residual connections. Even though it is a very simple model, it still achieves sufficient predictive power on tabular data. This layout also simplifies backpropagation, which can be helpful for the attack performance.

### 4.5.3 GANDALF

Gated Adaptive Network for Deep Automated Learning of Features (GANDALF) is a neural network architecture specifically designed for tabular data, introduced by Joseph et al. in 2022 [32]. We use this model in addition to MLP on tabular data. It builds on a custom component called the Gated Feature Learning Unit (GFLU), which is inspired by mechanisms from Gated Recurrent Units (GRUs) commonly used in time-series models.

At each layer, GFLU uses a mask to reweight the input features, effectively suppressing irrelevant information before further processing. This mask is normalized to form a probability distribution over input features. The result is a masked version of the original input, emphasizing the most relevant features for the current layer.

The GFLU then computes update and reset gates, similar to those in GRUs. These gates control how much of the previous layer's output should be preserved and how much new information should be incorporated.

This process allows each GANDALF layer to perform implicit feature selection, dynamically filtering out noise and focusing on informative signals. By doing this repeatedly across layers, GANDALF reduces overfitting and improves generalization. In addition, the model is efficient due to weight sharing across gates and the ability to exploit input sparsity.

One of GANDALF's advantages is interpretability: the learned feature masks at each layer can be inspected to reveal which original features the model deems important. Empirically, GANDALF achieves state-of-the-art performance on a range of tabular data benchmarks [32].

# 5

# Results

## 5.1 Image Domain: CelebA Dataset

Here we present the results of the PLG-MI attack on the CelebA data set, including comparisons to the corresponding setting in the original PLG-MI paper.

### 5.1.1 Target Model Training

**Table 5.1:** Target model test accuracies for the two different target models trained on $\mathcal{D}_{\text{priv}}$ as the first 1,000 identities of CelebA.

| **Metric** | **ResNet152** | **ResNet18** |
|---|---|---|
| Test accuracy (%) | $72.0 \pm 2.0$ | $68.0 \pm 2.0$ |

Table 5.1 shows the predictive performance of the 2 target models, where the deeper ResNet152 performs slightly better than the ResNet18 target model. Both models where trained to 100% training accuracy, indicating that the models are overfitted. The ResNet152 model was trained for 50 epochs and the ResNet18 model was trained for 100 epochs. The PLG-MI paper does not publish any target model statistics in this setting, so no comparison can be made to their setup. However, the MIBench paper reports a 93.7% test accuracy for a ResNet152 target model trained on 1,000 classes of $64 \times 64$ CelebA images (Table 9, Qiu et al. [14]). They specify that the top 1000 CelebA identities with the most images per identity are selected as $\mathcal{D}_{priv}$. This selection strategy may partly account for the discrepancy between their reported test accuracy and ours, as their approach mitigates class imbalance, see Figure A.2. This in combination with the larger data set achieved with their selection strategy (30,038 images vs. 21,152 images) is likely to contribute to the higher test accuracy compared to our dataset. The private data set used in the PLG-MI paper is reported to contain 30,027 images, indicating that their selection strategy is much closer to MIBench's $\mathcal{D}_{priv}$ compared to ours.

### 5.1.2 Attack Evaluation Metrics

Table 5.2 shows our results on CelebA in comparison to the results reported in Table 1 of the PLG-MI paper under the corresponding setup. Here, we can see that the attack is successful in terms of accuracy, on both ResNet152 and ResNet18 target

**Table 5.2:** Results of the PLG-MI attack on the CelebA dataset using two different target models and the results reported in the PLG-MI paper for ResNet152 as comparison [4] .

| Metric | ResNet152 | ResNet18 | PLG-MI paper: ResNet152 |
|---|---|---|---|
| Attack Accuracy (%) | 99.0 ± 1.0 | 98.0 ± 2.0 | 100.0 ± 0.0 |
| k-NN Distance | 1031.85 | 1119.06 | 1026.71 |
| FID | 33.42 | 35.36 | 23.22 |

models, similarly to previously reported results. Additionally, the k-NN distance is marginally lower in the PLG-MI results (1026.71 vs. 1031.85), and the FID score is notably better at 23.22 compared to the 33.42 observed in the current ResNet152 experiments. Because our accuracy and KNN-Distance metrics align with the results from the paper, it is likely that the discrepancy in FID is due to a difference in implementation. As mentioned above in 5.1.1, we use a different subset of CelebA as $\mathcal{D}_{priv}$ in comparison to the PLG-MI paper, likely explaining some of the difference. We also see that the attack on the ResNet18 target model yields slightly worse results compared to the ResNet152 case. This result is an indication that larger target models are more vulnerable for attack.

### 5.1.3 Analysis of Reconstructed Images

The results of the PLG-MI image reconstruction are shown in Figure 5.1 along with a corresponding example from the private data set $\mathcal{D}_{priv}$. Here, we manually chose that example based on what we deemed most similar to the reconstructed image. As seen in Figure 5.2, one reason that reconstruction can fail for a specific identity is due to high class variability (top row) and succeed due to lower class variability (bottom row). This figure also illustrates the implications of typical reconstruction attacks; it is easy to see the common features represented in the bottom identity, and more difficult to identify commonalities in the upper identity.

The results indicate that we have managed to successfully replicate the PLG-MI attack on CelebA images. The results are slightly different to the PLG-MI paper, as expected from the differences in setup of private data set and target model.

**Figure 5.1:** Comparison between generated (top row) and real (bottom row) images. Each column shows a matched pair based on their label. The generated images are displayed above one of their corresponding real images from $\mathcal{D}_{priv}$. The images are generated by an attack on a ResNet152 target model.

**Table 5.3:** Target model training accuracy and test accuracy when trained on the different settings of $\mathcal{D}_{priv}$.

| Train acc. (%) | Test acc. (%) | ID |
|---|---|---|
| 82.6 | 36.1 | 0 |
| 84.0 | 42.2 | 1 |
| 82.7 | 56.0 | 2 |
| 85.6 | 60.6 | 3 |
| 79.6 | 58.3 | 4 |

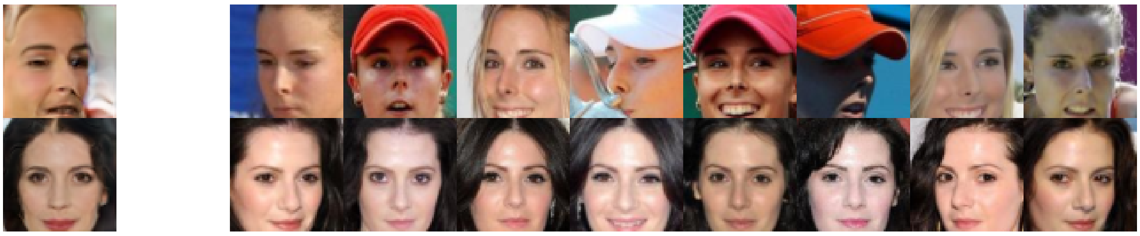


**Figure 5.2:** Comparison between fake and real images. Each row shows one fake image (left) followed by several real images of the same class (right). The first row corresponds to class 335, where the diversity among the real images (different lighting, poses, accessories) makes accurate reconstruction difficult. The second row corresponds to class 197, where the real images are visually homogeneous, facilitating easier and more consistent reconstruction of the individual's appearance. The images are generated by an attack on a ResNet152 target model.

## 5.2 Tabular Domain: MIMIC-IV Dataset

In this section we present the results from our attack on the MIMIC-IV Dataset. Because there are no previously published results in this setting, the comparisons will focus on evaluating in which setting the attack performs best.

### 5.2.1 Target Model Training

Table 5.3 shows the predictive performance of the 5 target models trained under different settings of $\mathcal{D}_{\text{priv}}$, as specified in Table 4.3. To reiterate, here each class corresponds to a specific diagnosis code, and not to a specific identity, as in section 5.1. The models trained with 100 and 50 classes selected, ID 0 and 1, respectively, show the largest generalization gaps and are therefore the most overfitted. The target models of ID 0-3 trained for 45, 37, 29 and 38 epochs, respectively. For the target models of ID 2 and 4, which are both trained on the same $\mathcal{D}_{\text{priv}}$, but with the MLP and GANDALF architectures, respectively, the GANDALF target model achieves better generalization at a lower training accuracy than its MLP counterpart. The GANDALF model also trained for longer; 97 iterations. The GANDALF evaluation models trained for all other settings displayed similar results; more training epochs and better generalization.

**Table 5.4:** Selected CTGAN training hyperparameters.

| Symbol | Value |
|---|---|
| `top_n` | 100 |
| $n_{\text{dis}}$ | 5 |
| $n_{\text{iter}}$ | 2,500 |
| `batch_size` | 500 |
| $\eta_{\text{gen}}$ | 2e-4 |
| $\lambda_{\text{gen}}$ | 1e-6 |
| $\eta_{\text{dis}}$ | 2e-5 |
| $\lambda_{\text{dis}}$ | 1e-7 |
| $\beta_1$ | 0.5 |
| $\beta_2$ | 0.9 |
| $\text{dim}_z$ | 256 |
| $\alpha_1$ | 1 |
| $\alpha_2$ | 1 |

### 5.2.2 Hyperparameters

The hyperparameters found to yield a converging and proper training loop can be seen in Table 5.4, which applies for all CTGAN models trained of setting 0 through 4. The CTGAN loss metrics over epochs for all settings can be seen in Section A.3, and the reason for keeping them fixed across settings is discussed.

Note that the Inversion loss scaling and conditioning loss scaling parameters were set to 1, effectively removing their impact on the attack. For the impact of $\alpha_1$ on attack performance, see ablation study in Section A.4.

### 5.2.3 Attack Evaluation Metrics

Table 5.5 shows the attack accuracy averaged across all evaluated classes. Notably, higher accuracy is observed for attack settings with fewer classes (ID 2-4, each with 20 classes), even though these models are less over-fitted. On the other hand, when considering two different attacks on the same $\mathcal{D}_{\text{priv}}$, setting 4 yields lower attack accuracy compared to setting 2, despite the only difference being the choice of target model, where the model in setting 4 exhibits less overfitting, as previously shown.

The mean shape and pair trends scores are also presented across all columns. That is, the mean shape and pair trends scores shown are for data that are non-conditionally generated for particular labels. Hence, this comparison assesses the general synthetic/reconstructed distributions of the trained CTGAN. In contrast to the attack accuracy results, better shape and pair trends scores were observed in settings with either more classes (IDs 0 and 1) or more features (ID 3); that is, settings where the dimension of the private data set was larger with either more rows or columns.

To evaluate the quality of the reconstructed from a class level aspect, column shape scores are computed for each column within each class. Specifically, for every class, a set of reconstructed samples is generated; conditioned on the class label; such that

**Table 5.5:** Performance metrics (mean ± standard deviation) over 10 evaluations for five configurations of $\mathcal{D}_{priv}$ (IDs 0-4), displaying both the prediction and distributional qualities of the reconstructed data. The metrics include attack accuracy, shapes score and pair trends score. For attack accuracy, 1000 samples is conditionally generated and evaluated for all classes. For test setting ID 0, two of the classes (98,99) were not separated when pseudo-labeling the public dataset; no public data points were assigned to these classes by the target model. These two classes were not evaluated when computing the accuracy for this setting. The shapes and pair trends scores are however evaluated on a reconstructed dataset with the same number of entries as the private, with no enforcement of an equal amount of samples in each class.

| **Attack Accuracy (%)** | **Shapes Score (%)** | **Pair Trends Score (%)** | **ID** |
|---|---|---|---|
| 52.56 ± 2.74 | 91.73 ± 0.01 | 83.94 ± 0.26 | 0 |
| 68.59 ± 2.93 | 91.44 ± 0.01 | 81.97 ± 0.62 | 1 |
| 82.13 ± 3.55 | 90.74 ± 0.03 | 82.40 ± 0.41 | 2 |
| 78.17 ± 5.36 | 92.12 ± 0.02 | 86.17 ± 0.08 | 3 |
| 75.76 ± 4.60 | 90.69 ± 0.03 | 79.54 ± 0.57 | 4 |

the number of samples matches that of the corresponding class in the private dataset. Since there are many columns, they are grouped by the variable/data type of that column and the mean is computed, for visualization purposes. In Figures 5.3 and 5.4, this evaluation is shown for test setting ID 2 and 3, respectively. It is clear that some reconstructed class distributions better match the private data class distributions, such as class 5, 8, 12, 19, to name a few. In general, the lowest scoring feature type is the discrete, and highest the categorical/boolean. The categorical/boolean features score higher in test setting 3 than in 2 on average, but the numerical features score lower.

Further, another informative comparison can be in analyzing the $n$ lowest scoring column shapes, in this case for the assessment that was performed to yield Table 5.5. Consider test settings with IDs 2 and 3, which are identical except for the number of features selected, 50 and 100, respectively. The additional features in setting 3 are predominantly categorical/boolean, as most of the remaining available features fall are of this type.

While the overall average score across all columns is higher for setting 3, as previously illustrated, the 8 lowest-scoring columns perform worse than those in setting 2, see Figures 5.5 and 5.6. This suggests that the inclusion of additional boolean columns, which individually contain less information when compared to other column types, may degrade the performance of the weakest column shapes, among which numerical columns; continuous and discrete; are disproportionately overrepresented.

### 5.2.4 Comparison of latent space optimization techniques

In this experiment we used parameters from ID 2. Here we compare the 2 proposed $\boldsymbol{z}$-optimization techniques described in sections 4.3.3 and 4.3.4, as well as randomiz-

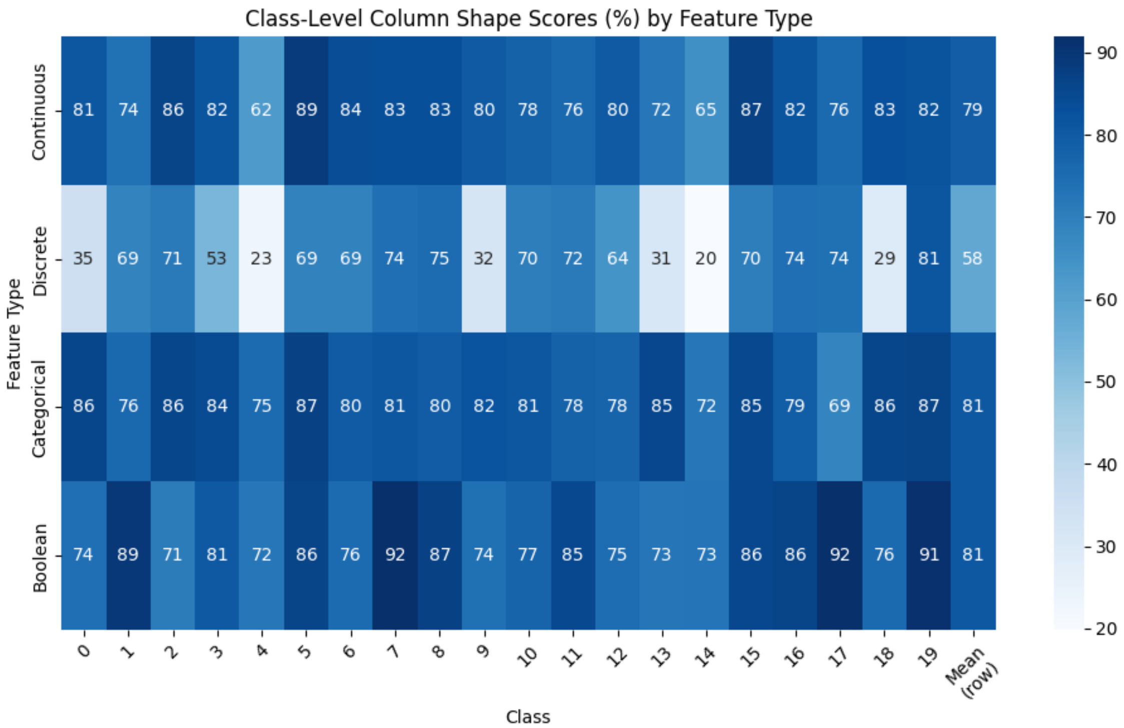



**Figure 5.3:** Class-level column shape scores distribution summary of setting ID 2, obtained by grouping features by data type (numerical, discrete, categorical, and boolean) and computing the mean within each group. Originally consisting of 50 individual features. The figure was reduced to 4 aggregated rows for improved interpretability. The last column is the mean computation of the rows, indicating an average score for each data type across all classes.

ing $\boldsymbol{z}$. The results in Table 5.6 show that there is no statistically significant difference between randomizing $\boldsymbol{z}$ and using the PLG-MI Gradient backpropagation optimization. Our suggested Bayesian approach show lower accuracy than random guessing. While both optimization methods fail to increase the attack accuracy, the PLG-MI gradient method manages to decrease the inversion loss in comparison to randomizing $\boldsymbol{z}$. This indicates that there is a lack of correlation between cross-entropy inversion loss and evaluation model accuracy in this setting.

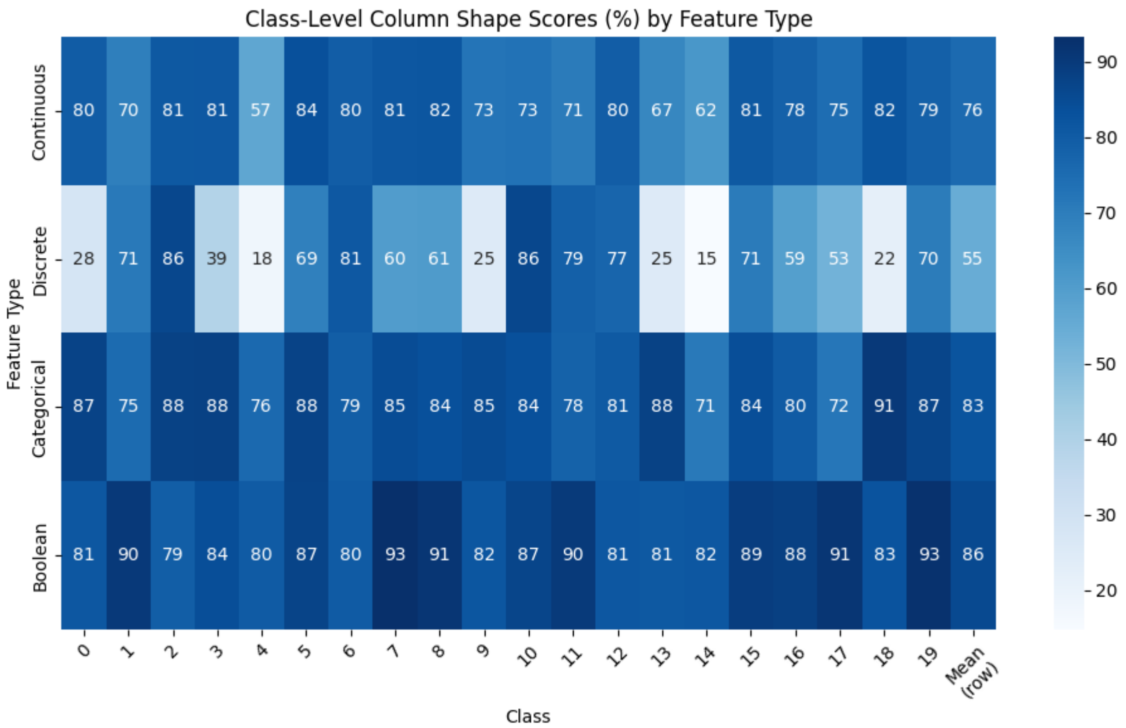



**Figure 5.4:** Class-level column shape scores of setting ID 3. Produced in the same way as specified in Figure 5.3.

**Table 5.6:** Evaluation accuracy for the three proposed optimization algorithms, and the cross entropy inversion loss after optimization, evaluated at the returned $\boldsymbol{z}^*$. Averaged over 5 runs.

| **Metric** | **PLG-MI Gradient** | **Bayesian** | **Random** $z$ |
|---|---|---|---|
| Attack Accuracy (%) | 85.40 ± 3.53 | 69.00 ± 8.00 | 84.80 ± 3.62 |
| Inversion loss (CE) @ $\boldsymbol{z}^*$ | 6.84 ± 1.47 | 12.14 ± 3.82 | 13.05 ± 5.82 |

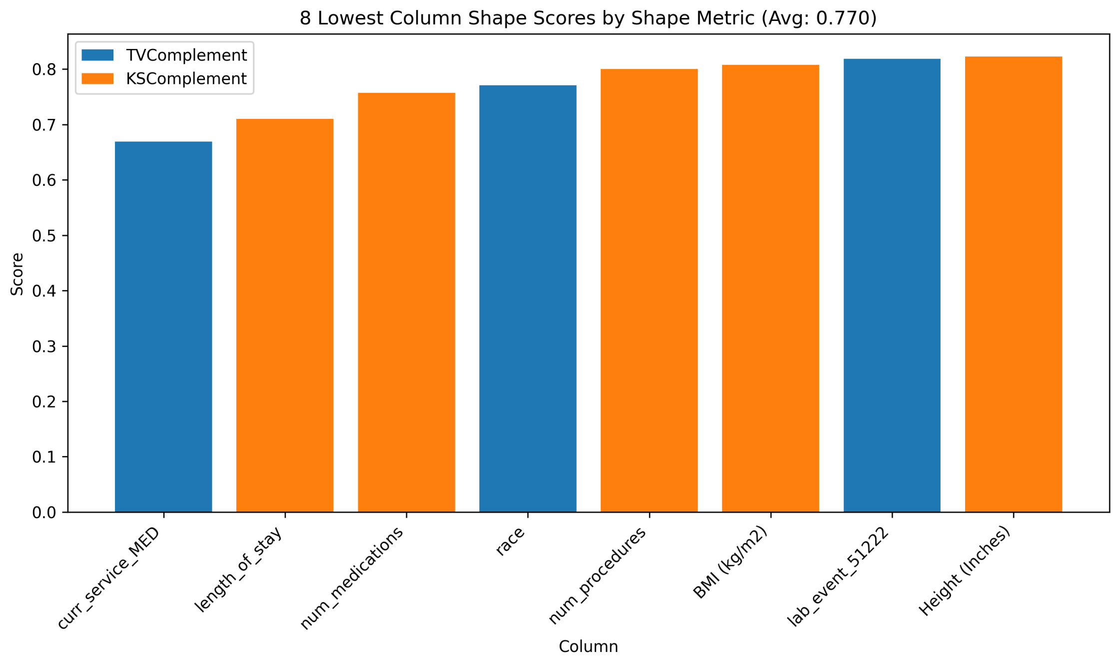



**Figure 5.5:** Column shape scores in terms of the complements of the Kolmogorov-Smirnov statistic and total variation distance for the 8 lowest scoring columns for experimental setting ID 2, with an average score displayed for the selected columns.

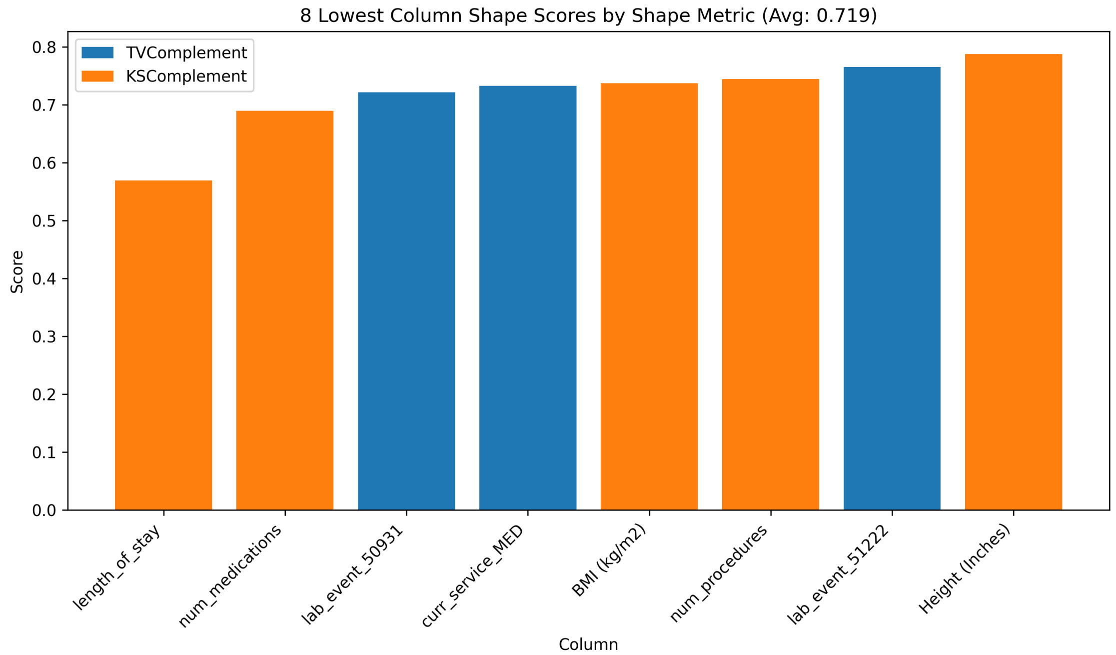



**Figure 5.6:** Column shape scores for the 8 lowest scoring columns for experimental setting ID 3, with an average score displayed for the selected columns.

# 6

# Discussion

How well do our results breach privacy? The purpose of Model Inversion attacks is to evaluate and highlight privacy risks in machine learning models. In this section we discuss the implications of the different metrics, results and conditions that we investigated. These discussions should give a wholistic picture of MI attacks on tabular data. Additionally, we suggest areas of further investigation which this thesis cannot fully address.

## 6.1 Image reconstruction evaluation

As seen in Table 5.2, we find that PLG-MI remains highly effective even when our private CelebA split is deliberately unbalanced. While our reconstruction quality metrics and attack accuracy are marginally lower than those reported in the original PLG-MI paper, which uses a near-perfectly balanced private data set, these degradations are modest and do not undermine the overall vulnerability.

Importantly, this unbalanced setup also provides a new perspective on attack robustness. In highly balanced regimes (as investigated in previous literature), each identity contributes roughly the same number of images, simplifying the target model task. In contrast, our distribution forces the attack to contend with minority identities that appear far less frequently in the training set, yet we still observe successful reconstruction.

Because our unbalanced CelebA experiment succeeds, we infer that PLG-MI can likewise deal with the imbalance of the MIMIC data set. In practice, many real-world tabular datasets exhibit a similar imbalance, which strengthens the case that the shown privacy risks extend beyond idealized, balanced benchmarks.

## 6.2 Tabular data reconstruction evaluation

Since reconstruction attacks on tabular data is a new methodology, we also propose new ways to evaluate the reconstructed data. These build on existing metrics for evaluating similarity in distributions of individual features such as Columns Shape Scores and Column Pair Trends.

Calculating these distribution-wide metrics can answer how well the reconstructed data matches the training data. This is a useful step towards a successful reconstruction attack, and has a similar reasoning behind it to image metrics such as

FID. However, to get a better idea of how the reconstructed samples match their corresponding class in the training data, we proposed the Class-Level Column Shape Scores by Variable Group heatmap, as for example shown in Figure 5.3. We believe that this type of comparison gives a clear insight into what types of features and which classes are well reconstructed.

Attack Accuracy is an applicable evaluation metric for both image and tabular data. Our results show that we have achieved high attack accuracy during reconstruction of the MIMIC data. This metric tells us that the assigned classes of the reconstructed samples matches with that of the evaluation model prediction. However, it does not say how closely our reconstructed samples correspond to the private data. Therefore, attack accuracy needs to be put into the context of the data set and classification task. On our MIMIC task, we investigated relatively few classes with relatively many samples per class, as compared to the CelebA image task. This means that correctly reconstructing a specific class corresponds to a less granular division of the private data set.

As shown in Table 5.3, the target model test accuracies on MIMIC are lower in comparison to the target model accuracies of CelebA (Table 5.1). This can imply that there is more class overlap in this setting, and that decision boundaries are harder to find, which can make an attack more difficult and have a negative effect on our results.

Further, we can from Table 5.5 see that the attack performance metrics deteriorated when using GANDALF as target model (ID 4) compared to using an MLP as target model. Recall that $\mathcal{D}_{\text{priv}}$ is the same for these two settings. The conclusion could be made that less over-fitting within the same setting implicates a weaker attack, but it is not straight-forward to state how much of this deterioration is due to the different model architectures of target and evaluation model and how much is due to the fact that the GANDALF model is less over-fitted.

Another consideration concerns the evaluation of the column pair trends, for which Pearson's correlation coefficient was used for all numerical comparisons. It is important to highlight that this comparison only captures linear correlations. Given the large number of possible pairwise comparisons, it is not feasible to manually inspect and identify which pairs may exhibit non-linear dependencies. As a result, the column pair trends score can be considered a lower-bound estimate: reconstructed data may accurately reflect non-linear trends in the private data, yet still receive a low score due to the limitations of the linear correlation metric.

In contrast to the CelebA dataset, interpreting the results on the MIMIC dataset is more challenging due to the limited prior work on MI reconstruction attacks in the tabular data domain. The lack of established benchmarks or expectations makes it difficult to contextualize the results. Nonetheless, a comparative observation can be made: the MIMIC dataset appears more difficult to attack, as evidenced by the lower attack accuracy despite using fewer classes than in the CelebA experiments. The extent to which this is attributable to the raw nature of the MIMIC dataset, insufficient pre-processing and feature engineering, or the inherent complexities of tabular data; such as high intra-class variability; remains unspecified.

In the image domain, an approximate reconstruction can still be clearly identifiable. For tabular data, it is highly context-specific if the reconstruction can reveal sensitive information. In the MIMIC-IV example, accurate reconstruction of height and weight features can be enough to identify an individual for an adversary with the right prior knowledge.

Our constructed diagnosis classification task contains too many individuals per diagnosis code to make accurate inference about any individual patient. This is also in line with the terms and conditions of the use of MIMIC-IV. However, Figure 5.3 shows that we have achieved accurate reconstruction of the class-conditional distributions.

## 6.3 Individual vs. class typical reconstruction

Even though the PLG-MI attack is a typical reconstruction attack, we have used some comparisons to specific private data points in our evaluation, such as comparing images in Figure 5.1 or Gower's distances in Table A.2. These results show that at some samples of the reconstructed data are very closely aligned with some samples of the private data, in both the image and tabular domain. This indicates that the privacy has been breached for those private samples, because the adversary now has an accurate reconstruction of them. By extension, this means that the target model itself leaks private data, and therefore can be considered insecure.

However, the difficulty from the adversary's perspective lies in identifying which of the reconstructed samples are closest to real private samples—that is, determining which samples represent true individual reconstructions as opposed to class-typical representations. Since the attack does not inherently distinguish between these two, or rather aims for the class-typical reconstruction, some additional knowledge or tool is necessary for the adversary to be able to identify these individual reconstructions.

## 6.4 Optmization phase of PLG-MI on tabular data: limitations and adaptations

In our implementation, our latent space optimization fails to improve the attack results compared to using random noise, as seen in Table 5.6. The optimization methods investigated show reductions in inversion loss, but this does not seem to translate to increased accuracy in this setting.

We believe that there is a disconnect between cross-entropy loss as defined in the current gradient-based $\boldsymbol{z}$-optimization algorithm, and attack accuracy. This might be related to the task and setting itself, for example that the target model used does not reach 100% training accuracy, and thus less overfitted. It can also be because of the inherent mechanics of CTGAN. A contributing factor is likely that due to the method used, as described in algorithm 5, the cross-entropy loss is measured by comparing the target $y$ to a $y_{pred}$ produced by using the combined $f_{inv}$ conversion between $G$ and $T$, while the accuracy evaluation is evaluated on rows of reconstructed tabular data, which is required to utilize $f_g^{-1}$ and $f_T$. This result can therefore indicate a mismatch between the two transformation methods.

We suspect that the core issue lies in a lack of correlation between the cross-entropy objective used during gradient-based optimization of the latent vector $\boldsymbol{z}$ and the attack accuracy metric in this setting. Concretely, Algorithm 5 minimizes

$$\mathcal{L}^{\mathrm{CE}}(\boldsymbol{z}, y) = -\sum_i y_i \log T\Big(f_{\mathrm{inv}}(G(\boldsymbol{z}, y))\Big)_i,$$

where $f_{\mathrm{inv}}$ is the direct tensor conversion. In contrast, our accuracy is evaluated on reconstructed tabular records, which requires the use of $f_G^{-1}$ and $f_T$, then classifying that reconstructed record with $T$. This can indicate a mismatch between the two methods—one via $f_{\mathrm{inv}}$ and one via $f_G^{-1}$, $f_T$. The results might also be amplified by the fact that our target model never fully overfits (does not reach 100% training accuracy) which might create an effective upper limit for the evaluation accuracy, already reached by the training phase. There can also be differences of mechanics of CTGAN in comparison to the generator utilized in the image variant of PLG-MI, which might create a latent space coupling that isn't suited for gradient-based optimization.

## 6.5 Medical data ethical considerations

When working with medical data, including the MIMIC-IV dataset, special consideration has to be made not to reveal any sensitive information about the underlying data. Although MIMIC-IV is a de-identified dataset, the risk of re-identification through techniques such as model inversion attacks, remains a concern. These attacks can potentially reconstruct sensitive patient information from model outputs, posing significant privacy risks.

To mitigate these risks, we have ensured that our research does not expose any individual data points from the original dataset. All analyses are conducted on synthetic or aggregated data, and results are presented in a manner that precludes the possibility of re-identifying individuals.

## 6.6 Attack development considerations - developing stronger attacks

The development of new model inversion attacks raises a critical ethical question: Are we creating new vulnerabilities or uncovering existing ones? We argue that demonstrating the feasibility of such attacks reveals pre-existing weaknesses in machine learning models, particularly concerning the inadvertent memorization of sensitive training data.

By exposing these vulnerabilities, we aim to provoke ideas about new privacy-preserving techniques and encourage the implementation of safeguards against potential data breaches. For example, as seen in Figure 5.5 and 5.6, the inclusion of many additional easy-to-attack boolean features seem to indicate a direct deterioration of the attack on the more difficult features. This could motivate elaborate inclusion of non-privacy-leaking boolean features as a defensive measure. This proactive approach aligns with the broader goal of enhancing the security of machine learning models.

Furthermore, we also show in which settings the attacks are not successful. By extension, these settings are therefore more protected against attacks. Concretely, by rigorously evaluating the conditions under which inversion attempts repeatedly fail, we can treat these “attack-resistant” regimes as de facto defenses. As an example, we have found that tabular models that transform their input from structured to unstructured data formats, and do not provide their transformations to the adversary, are essentially resistant to the PLG-MI attack.

## 6.7 Further investigation

This section highlights observations and limitations that could be relevant for future works.

### 6.7.1 Top- $k$ and $n$ Selection

One potentially important detail in our attack setups is the fixed choice of top-n for pseudo-labeling. This parameter determines the maximum number of pseudo-labels assigned by the target model to each target class within the public data. However, it does not guarantee a minimum number of annotations, nor does it account for the quality of those annotations; selection is based solely on the ranking of predicted class probabilities, not the actual confidence scores. Therefore, this approach may not generalize well across different experimental conditions, particularly when the model is uncertain or when class distributions are imbalanced and data for minority classes is sparse. Perhaps a dynamic selection of this parameter is possible which depends on the distributions of the pseudo-labels, as an alternative.

Concretely, in our experiments, we considered three different settings by selecting the top-$k$ most frequent classes from the private dataset, with $k$ = 100, 50, and 20. As previously presented, the last two classes when choosing $k$ = 100 were not assigned *any* pseudo-labels. Naturally, this motivated to not proceed with auditing more than 100 classes. This indicates with very high possibility that the number of labels in the previous, nearby classes were not 100, or much worse; just a few. Here, a minimal amount of labels can be enforced, and n can be set to 0 if this criteria is not met, and the class be excluded from the attack. If labels on the other hand are abundant and many more than n, it can be increased to retain as much of the public data as possible. However, this would accentuate the imbalances that are already present.

### 6.7.2 Feature Selection

As the feature selection method for this thesis consisted of the standard mean decrease in impurity feature importance method, there are some implications of the quality of the selected features. This particular method does have some well-known issues. For instance, if the random forest model used to select the features is weak and if there are correlated features present can both lead to deteriorated feature selection quality [30]. Further, there exists a selection bias towards numerical and high cardinality features. Although all of these issues are indeed present in our particular dataset to some extent, we opted for this solution as it is highly effective and straightforward to implement.

As a proposal for future works, more thorough analysis can be considered for the dataset constructed for our particular classification task to ensure the quality of the selected features, one example being the permutation feature importance selection method.

### 6.7.3 Hyperparameter Tuning

Many hyperparameters across the various experimental settings were intentionally kept at default values to maintain consistency and reduce complexity in tuning. For example, in the case of CTGAN, parameters such as the learning rate ($\eta_{\text{gen}}$), weight decay ($\lambda_{\text{gen}}$), batch size and others, were all set to their default configurations. Additionally, the weighting factors $\alpha_1$ and $\alpha_2$ used in the loss function were both set to 1, effectively neutralizing their influence in the combined loss. This stands in contrast to prior work in the image domain, where, for instance, $\alpha_1$ was set to 0.2 to adjust the loss balance. These neutral choices could suggest that there is significant room for improvement through more targeted hyperparameter tuning, which could potentially enhance the effectiveness of an attack. The *Adam* optimizer used should allow for a wider range of initial learning rates, due to the adaptive nature of *Adam*, and can therefore mean that the current initial values of those parameters are sufficient.

### 6.7.4 Tabular Transformations and Backpropagation Difficulty

Do common tabular data transformations inherently make backpropagation more difficult, and thereby make the tabular data setting more challenging to attack? We've found that transformations between structured and unstructured formats can block gradient flow for an adversary, and that these transformations might introduce discontinuities or non-linearities that make it difficult to bypass them, as described in section 4.3.3. A detailed analysis is needed to determine how these transformations alters the Model Inversion landscape for tabular data, when certain transformations are possible to bypass, and if "attack-aware" preprocessing can be designed to prohibit reconstruction attacks and therefore increase privacy.

### 6.7.5 Realism of Target-Model Transformations

Is using the target model's fitted transformations (for example, normalizations computed on the private data) a realistic threat scenario, as we've employed it for bypassing format transformations? If an adversary can obtain these transformations, can access to such parameters be used to mount a stronger attack by using properties of the transformations as a statistical prior? In the case when these transformations are not directly available, to what extent can an attacker approximate them from public data? Investigating strategies for estimating or alternating between transformation parameter inference and input reconstruction will clarify the gap between idealized white-box attacks and more constrained practical settings.

As a further method improvement, assuming that the entire target model transformations $f_T$ are available to the adversary, is it possible to adapt the CTGAN architecture to use the same transformations $f_G = f_T$, instead of fitting the generator transformations to the public data? In that case, the bypassing transformation is equal to $f_{\text{inv}} = f_T \circ f_G^{-1} = \mathbb{I}$ (the identity function). This would eliminate the complexities associated with mismatched or approximated tabular transformations discussed earlier.

### 6.7.6 Adaptation of Black-Box Attacks to Tabular Data

Since we have established that white-box attacks are feasible on tabular data under certain settings, how well do black-box inversion methods adapt to this modality? We have focused a large part of this paper focusing on the issues and workarounds with gradient access and backpropagation of the tabular target models. Since black-box attacks do not rely on these steps, it might show a more relevant use case on tabular data in comparison to the image domain.

Can black-box attacks be successfully applied to both tree-based ensembles (e.g. random forests, gradient-boosted trees) and neural network models trained on tabular data? Decision-tree splits are inherently non-differentiable, which rule out white-box attacks.

### 6.7.7 Attack Performance Under Distributional Shift

In this paper we have assumed that a large, disjoint subset of the same dataset is available to the adversary. Since tabular data is often structurally unique from one dataset to another, how can inversion attacks be performed when public data from the same distribution is small or unavailable? Future work should examine whether reconstruction is possible for small in-distribution public data sets, and explore domain-adaptation techniques for surrogates to align out-of-distribution data sources.

# 7
# Conclusion

This thesis has studied how the PLG-MI attack can be adapted to attack tabular data DNN classifiers. This is to the best of our knowledge the first time a modern Model Inversion attack has been adapted from the image domain to the tabular domain. We have found that the PLG-MI attack generalizes to tabular data, with up to 82% evaluation accuracy on our MIMIC-IV diagnosis classification task. The distribution of the reconstructed tabular data converges towards the distribution of the private data and the generator is generating samples to the correct classes to the extent shown in the results.

We have reproduced the image-domain results of the original PLG-MI paper, visualized the implications of class-typical reconstruction, and verified that the attack functions under class imbalance, which is a discussion that were omitted from the paper, but is important especially in the context of tabular data.

To evaluate the reconstruction of tabular data, we have relied on previous metrics, such as the model inversion specific attack accuracy, and tabular specific column shape and column trend scores, which are proper even in assessing MI attacks with distributional shifts present, as both location and shapes of the distributions are captured. In addition to this, we have introduced Class-level column shape scores to identify how well our reconstruction reproduces the distributions of each class. We have used Gower's distance as a dissimilarity metric to find the best reconstructed tabular samples.

We have also identified key properties of machine learning algorithms on tabular data that complicate or inhibit different aspects of Model Inversion attacks in the tabular domain. Namely, the backpropagation steps of the PLG-MI attack, where target model gradients are utilized, are not possible when the target applies data format conversions, for example transformations between DataFrame format and tensor format as discussed in section 4.3.3. Our adaptation therefore bypasses these transformations by mimicking encoding steps and normalizations using only tensor operations. Doing this assumes that additional information, specifically transformations fitted to the private data, is available to the adversary. In our specific example, this $\boldsymbol{z}$-optimization procedure reduces inversion loss but does not significantly increase attack accuracy compared to random $\boldsymbol{z}$. Since the $\boldsymbol{z}$-optimization step of the PLG-MI attack relies on a deterministic generation process, the non-deterministic Gumbel-Softmax activation function of CTGAN should therefore be replaced for meaningful analysis to be conducted here. In our case, we simply replaced this with a regular softmax.

Backpropagation is also not as straight-forward when the target model is not a neural network model, for example XGBoost models. This is relevant for the tabular domain because tree-based models can often match or exceed the capabilities of SOTA neural networks, and are therefore utilized. This in contrast to image classification where SOTA neural networks dominate classical machine learning models.

Model Inversion Attacks demonstrate a major privacy threat towards neural network models trained on sensitive data. It is already established that these attacks are highly effective on image models. In this thesis, we have shown that the PLG-MI attack can be adapted to tabular data, and that therefore MI Attacks are applicable in the tabular domain, aspiring to spur further works on the topic. We have also delineated in which settings model inversion attacks can be hindered. Investigating both when these attacks work, and when they do not, create insight that allow for more robust defenses against MI attacks in the future.

# Bibliography


[1] Reza Shokri et al. "Membership Inference Attacks Against Machine Learning Models". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 3–18. DOI: 10.1109/SP.2017.41.

[2] European Parliament and Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council. of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. OJ L 119, 4.5.2016, p. 1–88. May 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504 (visited on 05/26/2025).

[3] Yugeng Liu et al. *ML-Doctor: Holistic Risk Assessment of Inference Attacks Against Machine Learning Models*. 2021. arXiv: 2102.02551 [cs.CR]. URL: https://arxiv.org/abs/2102.02551.

[4] Xiaojian Yuan et al. *Pseudo Label-Guided Model Inversion Attack via Conditional Generative Adversarial Network*. 2023. arXiv: 2302.09814 [cs.CV]. URL: https://arxiv.org/abs/2302.09814.

[5] Michael Veale, Reuben Binns, and Lilian Edwards. "Algorithms that remember: model inversion attacks and data protection law". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 376.2133 (Oct. 2018), p. 20180083. ISSN: 1471-2962. DOI: 10.1098/rsta.2018.0083. URL: http://dx.doi.org/10.1098/rsta.2018.0083.

[6] Yael Itzhakev et al. *Addressing Key Challenges of Adversarial Attacks and Defenses in the Tabular Domain: A Methodological Framework for Coherence and Consistency*. 2024. arXiv: 2412.07326 [cs.LG]. URL: https://arxiv.org/abs/2412.07326.

[7] Matthew Fredrikson et al. "Privacy in pharmacogenetics: an end-to-end case study of personalized warfarin dosing". In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. SEC'14. San Diego, CA: USENIX Association, 2014, pp. 17–32. ISBN: 9781931971157.

[8] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA: Association for Com-

puting Machinery, 2015, pp. 1322–1333. ISBN: 9781450338325. DOI: 10.1145/2810103.2813677. URL: https://doi.org/10.1145/2810103.2813677.

[9] Sayanton V. Dibbo. “SoK: Model Inversion Attack Landscape: Taxonomy, Challenges, and Future Roadmap”. In: *2023 IEEE 36th Computer Security Foundations Symposium (CSF)*. 2023, pp. 439–456. DOI: 10.1109/CSF57540.2023.00027.

[10] Si Chen et al. *Knowledge-Enriched Distributional Model Inversion Attacks*. 2021. arXiv: 2010.04092 [cs.LG]. URL: https://arxiv.org/abs/2010.04092.

[11] Yixiang Qiu et al. *A Closer Look at GAN Priors: Exploiting Intermediate Features for Enhanced Model Inversion Attacks*. 2024. arXiv: 2407.13863 [cs.CV]. URL: https://arxiv.org/abs/2407.13863.

[12] Ngoc-Bao Nguyen et al. *Re-thinking Model Inversion Attacks Against Deep Neural Networks*. 2023. arXiv: 2304.01669 [cs.LG]. URL: https://arxiv.org/abs/2304.01669.

[13] Ngoc-Bao Nguyen et al. *Label-Only Model Inversion Attacks via Knowledge Transfer*. 2023. arXiv: 2310.19342 [cs.LG]. URL: https://arxiv.org/abs/2310.19342.

[14] Yixiang Qiu et al. *MIBench: A Comprehensive Benchmark for Model Inversion Attack and Defense*. 2024. arXiv: 2410.05159 [cs.CV]. URL: https://arxiv.org/abs/2410.05159.

[15] Bernhard Mehlig. *Machine Learning With Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021. DOI: 10.1017/978110849493.

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[17] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax*. 2017. arXiv: 1611.01144 [stat.ML]. URL: https://arxiv.org/abs/1611.01144.

[18] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, 2006. ISBN: 978-0-471-24195-9.

[19] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. *Why do tree-based models still outperform deep learning on tabular data?* 2022. arXiv: 2207.08815 [cs.LG]. URL: https://arxiv.org/abs/2207.08815.

[20] Shuhei Watanabe. *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. 2023. arXiv: 2304.11127 [cs.LG]. URL: https://arxiv.org/abs/2304.11127.

[21] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: https://arxiv.org/abs/1406.2661.

[22] Lei Xu et al. *Modeling Tabular data using Conditional GAN*. 2019. arXiv: 1907.00503 [cs.LG]. URL: https://arxiv.org/abs/1907.00503.

[23] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: 1706.08500 [cs.LG]. URL: https://arxiv.org/abs/1706.08500.

[24] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV]. URL: https://arxiv.org/abs/1512.00567.

[25] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. "The Synthetic data vault". In: *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. Oct. 2016, pp. 399–410. DOI: 10.1109/DSAA.2016.49.

[26] Ouxiang Li et al. *Model Inversion Attacks Through Target-Specific Conditional Diffusion Models*. 2024. arXiv: 2407.11424 [cs.CV]. URL: https://arxiv.org/abs/2407.11424.

[27] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. Dec. 2015.

[28] Alistair E. W. Johnson et al. "MIMIC-IV, a freely accessible electronic health record dataset". In: *Scientific Data* 10.1 (2023), p. 1. ISSN: 2052-4463. DOI: 10.1038/s41597-022-01899-x. URL: https://doi.org/10.1038/s41597-022-01899-x.

[29] Manu Joseph. *PyTorch Tabular: A Framework for Deep Learning with Tabular Data*. 2021. arXiv: 2104.13638 [cs.LG].

[30] Leo Breiman. *Random Forests*. Published in Machine Learning, Volume 45, Issue 1, pp. 5–32. 2001. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[31] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[32] Manu Joseph and Harsh Raj. *GANDALF: Gated Adaptive Network for Deep Automated Learning of Features*. 2024. arXiv: 2207.08548 [cs.LG]. URL: https://arxiv.org/abs/2207.08548.

# A

# Appendix 1

## A.1 MIMIC-IV Data preparation

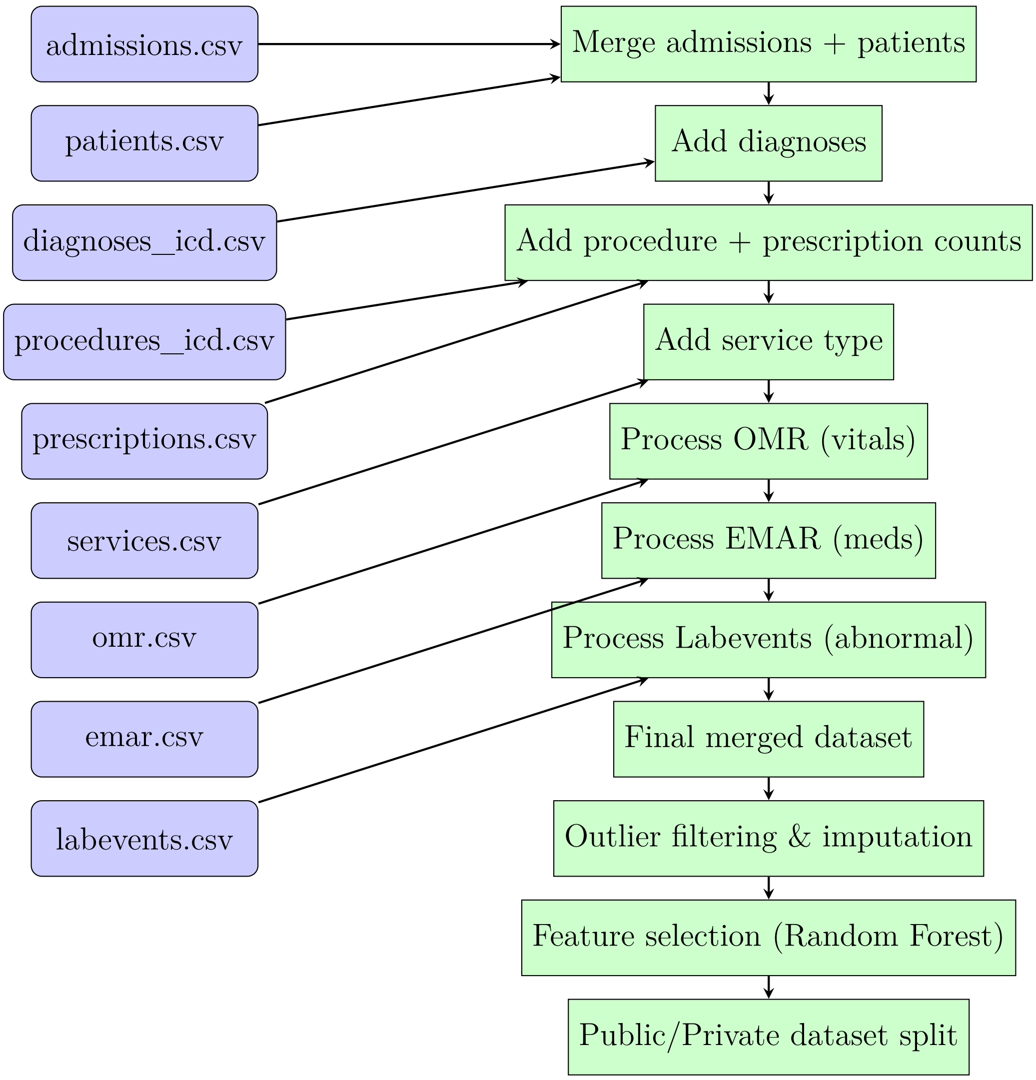



**Figure A.1:** Data pipeline for constructing the final dataset from MIMIC-IV tables. Raw CSVs are merged and processed in sequence to extract demographic, diagnostic, procedural, medication, observation, and lab features. Everything is joined on hospital admission id.

## A.2 CelebA Private Data Distribution

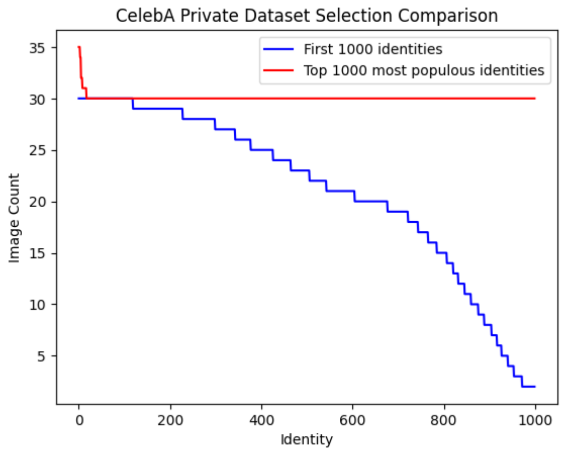


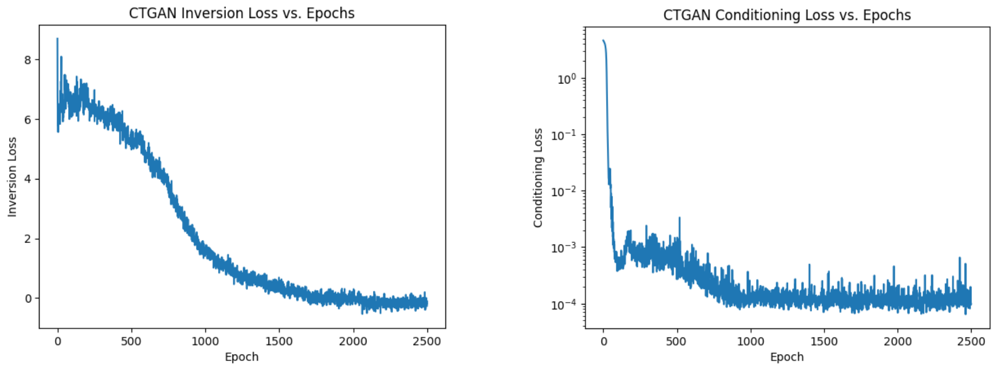

**Figure A.2:** The difference in per class image counts for 2 different 1,000 class subsets of the CelebA dataset, depending on if the first 1,000 identities are selected or if the 1,000 most populous identities are selected. The dataset represented by the blue line represents contains 21,152 images in total and the dataset represented by the red line contains 30,038.

Figure A.2 illustrates the distributional difference of our selection of $\mathcal{D}_{priv}$, and selecting the most populous identities for $\mathcal{D}_{priv}$ as done in the MIBench paper. Table A.1 shows test accuracies of target models trained on the balanced $\mathcal{D}_{priv}$. This shows higher results in comparison to table 5.1, but still slighly lower ResNet152 results in comparison to the 93.7% test accuracy reported in the MIBench paper. This discrepancy is likely due to differences in hyperparameter tuning.

**Table A.1:** Target model accuracies for the two different target models trained on the balanced, 30.038 image 1.000 class selection of $\mathcal{D}_{priv}$.

| **Metric** | **ResNet152** | **ResNet18** |
|---|---|---|
| Test accuracy (%) | 85.0 ± 2.0 | 73.0 ± 2.0 |

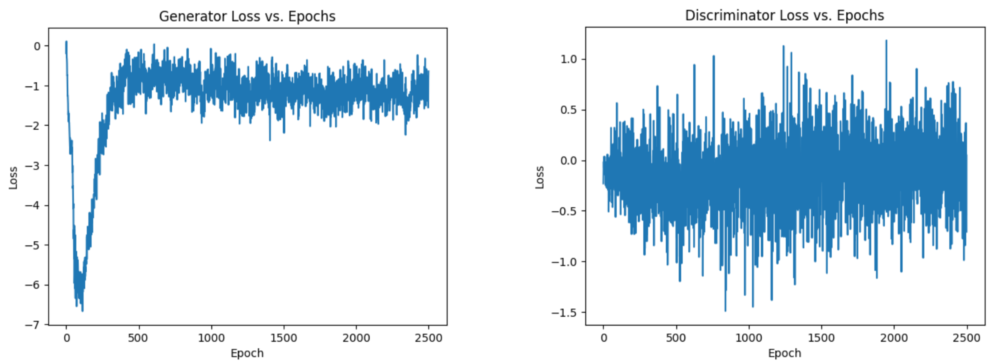
## A.3 CTGAN Training




**(a)** Inversion loss over epochs.

**(b)** Conditioning loss over epochs in logarithmic scale.

**Figure A.3:** Inversion and conditioning losses over epochs for setting ID 0.



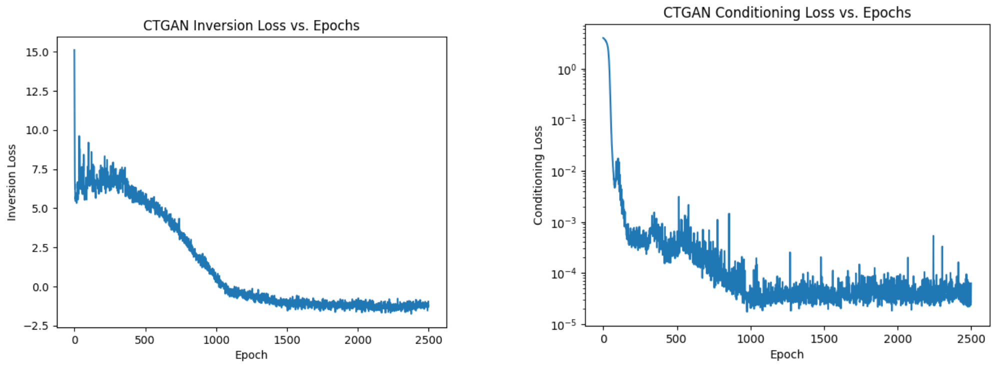

**(a)** Generator loss over epochs.

**(b)** Discriminator loss over epochs in logarithmic scale.

**Figure A.4:** Generator and discriminator losses over epochs for setting ID 0.

Figures A.3 - A.12 show various loss metrics over epochs for the CTGAN models trained corresponding to settings ID 0 through 4, whose attack metrics was evaluated in Section 5.2.3. Notably, the training dynamics for all of the models were similar, which is why the hyperparameters were kept constant for these models. The one exception was, as previously mentioned, that of setting ID 3 where the number of epochs were set to 5000. By inspecting the figure for this setting, this decision is made clear. Further, since it was shown in the PLG-MI paper that as long as a sufficiently large top-n is selected, this hyperparameter has minimal impact, motivating the fixed selection for this particular hyperparameter.

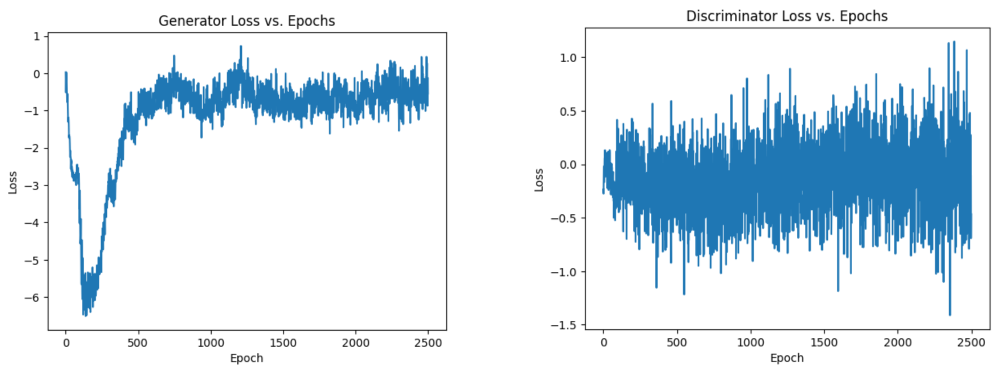
Although hyperparameter tuning to optimize attack metrics for each attack setting can be motivated, making comparison fairer, this time investment would hinder and restrict our main focus of analysis; the investigation of the generalization and feasibility of model inversion attacks to unexplored domains under different settings.




**(a)** Inversion loss over epochs.

**(b)** Conditioning loss over epochs in logarithmic scale.

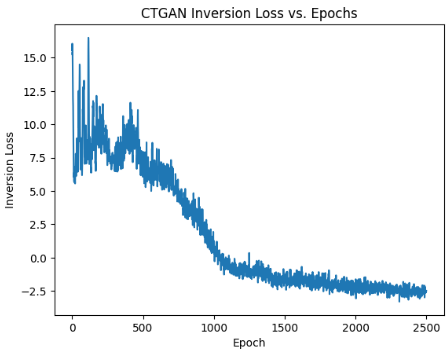
**Figure A.5:** Inversion and conditioning losses over epochs for setting ID 1.



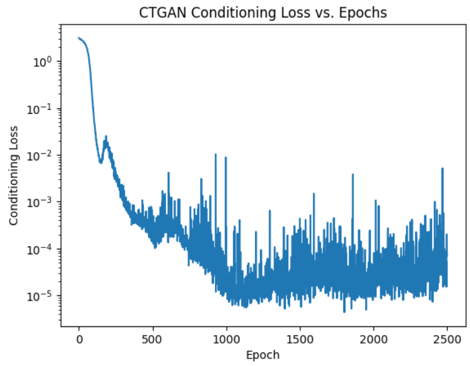

**(a)** Generator loss over epochs.

**(b)** Discriminator loss over epochs in logarithmic scale.

**Figure A.6:** Generator and discriminator losses over epochs for setting ID 1.

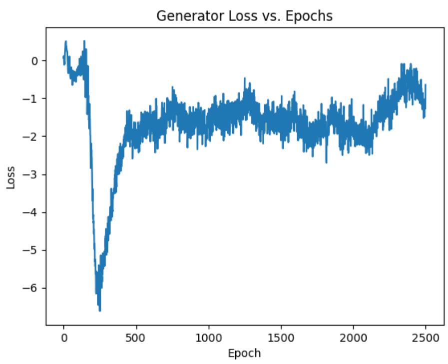



**(a)** Inversion loss over epochs.

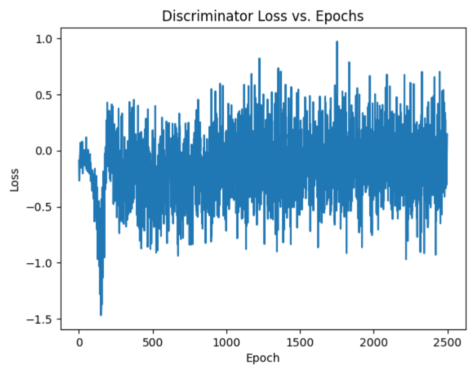



**(b)** Conditioning loss over epochs in logarithmic scale.

**Figure A.7:** Inversion and conditioning losses over epochs for setting ID 2.

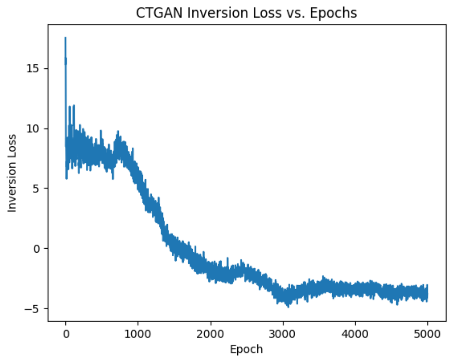



**(a)** Generator loss over epochs.

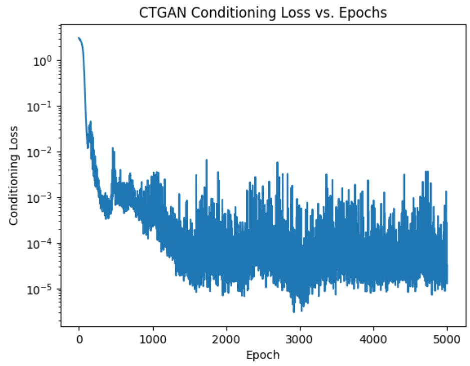



**(b)** Discriminator loss over epochs in logarithmic scale.

**Figure A.8:** Generator and discriminator losses over epochs for setting ID 2.

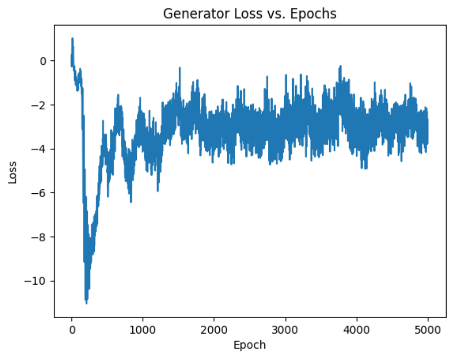



(a) Inversion loss over epochs.

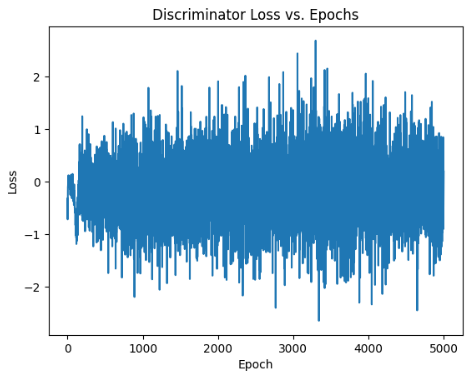



(b) Conditioning loss over epochs in logarithmic scale.

**Figure A.9:** Inversion and conditioning losses over epochs for setting ID 3.

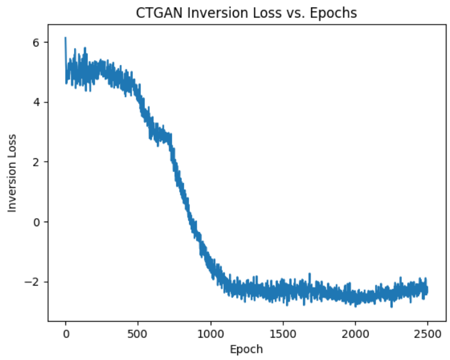



(a) Generator loss over epochs.

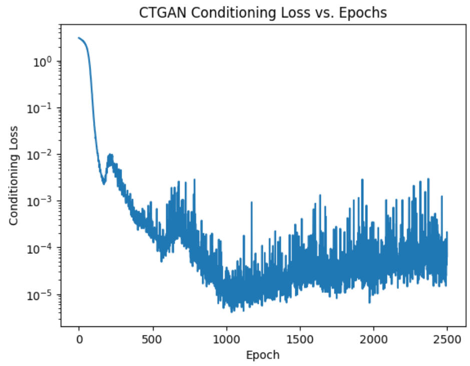



(b) Discriminator loss over epochs in logarithmic scale.

**Figure A.10:** Generator and discriminator losses over epochs for setting ID 3.

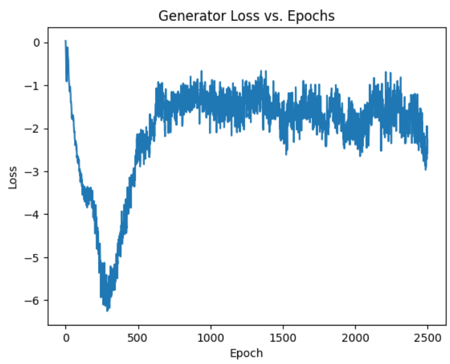



**(a)** Inversion loss over epochs.

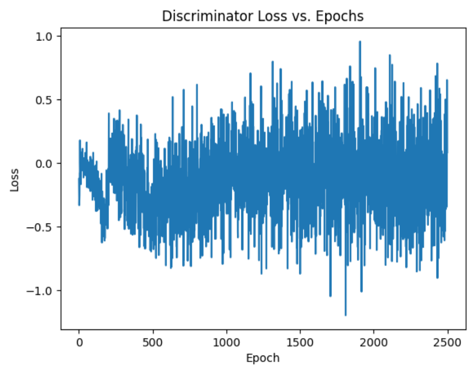



**(b)** Conditioning loss over epochs in logarithmic scale.

**Figure A.11:** Inversion and conditioning losses over epochs for setting ID 4.




**(a)** Generator loss over epochs.




**(b)** Discriminator loss over epochs in logarithmic scale.

**Figure A.12:** Generator and discriminator losses over epochs for setting ID 4.

## A.4 Ablation Study

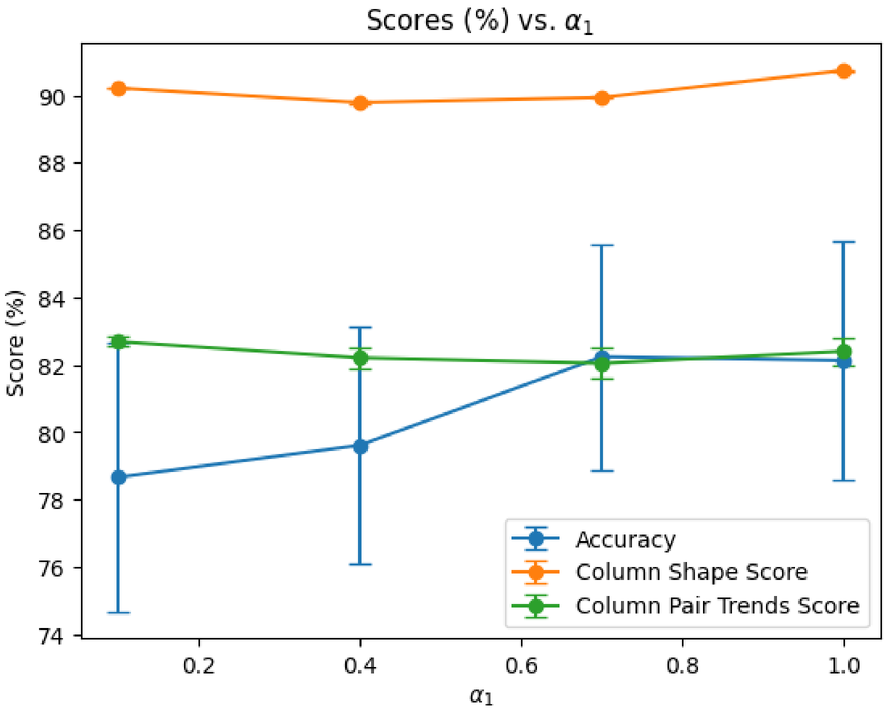



**Figure A.13:** Impact of $\alpha_1$ on Attack Accuracy, average Column Shape Score and Column Pair Trends Score across all columns.

Figure A.13 highlights the impact of of $\alpha_1$ on the different attack metrics, for some values in the interval [0,1]. In contrast to the PLG-MI paper, here we see best results when selecting $\alpha_1 = 1$. It is important to note that this parameter is not bound to the interval investigated.

## A.5 MIMIC-IV Gowers Distance evaluation

**Table A.2:** The difference of values between the closest reconstructed rows to the private data for each class (diagnosis code), measured by Gower's distance. For the categorical feature, the category is not listed, but instead a boolean is displayed if it was correctly reconstructed. All remaining features, which are either categorical or boolean, were correctly reconstructed for all listed entries.

| Class | BMI (kg/m2) | Height (Inch) | num_meds | num_proc | len_of_stay | race |
|---|---|---|---|---|---|---|
| 0 | 0.44 | 0.05 | 0 | 0 | 0 | True |
| 1 | 0.02 | 0.11 | 11 | 0 | 2 | True |
| 2 | 0.68 | 1.51 | 1 | 0 | 3 | True |
| 3 | 0.06 | 0.06 | 4 | 0 | 1 | True |
| 4 | 0.21 | 0.07 | 9 | 0 | 1 | True |
| 5 | 9.40 | 2.03 | 4 | 0 | 5 | True |
| 6 | 0.29 | 0.06 | 11 | 1 | 0 | False |
| 7 | 0.07 | 0.03 | 2 | 0 | 1 | True |
| 8 | 0.05 | 0.03 | 1 | 0 | 0 | False |
| 9 | 1.30 | 0.89 | 1 | 0 | 1 | True |
| 10 | 4.83 | 2.74 | 11 | 1 | 2 | True |
| 11 | 0.25 | 0.04 | 20 | 1 | 4 | True |
| 12 | 14.87 | 0.69 | 9 | 0 | 1 | True |
| 13 | 0.02 | 0.01 | 0 | 0 | 0 | True |
| 14 | 0.09 | 0.08 | 0 | 0 | 1 | True |
| 15 | 3.88 | 0.07 | 17 | 1 | 2 | True |
| 16 | 1.27 | 0.62 | 17 | 0 | 1 | True |
| 17 | 1.18 | 0.57 | 1 | 0 | 2 | True |
| 18 | 0.20 | 0.01 | 5 | 0 | 1 | True |
| 19 | 0.11 | 0.05 | 4 | 2 | 4 | False |

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se