CLUES: Collusive Theft of Conditional Generative Adversarial Networks

CLUES: Collusive Theft of Conditional Generative Adversarial Networks

Simon Queyrut, Valerio Schiavoni, Lydia Y. Chen, Pascal Felber and Robert Birke*
Institute of Computer Science (IIUN)
University of Neuchâtel, Neuchâtel, Switzerland
first.last@unine.ch
*University of Turin, Turin, Italy
robert.birke@unito.it

***Abstract*—Conditional Generative Adversarial Networks (cGANs) are increasingly popular web-based synthesis services accessed through a query API, *e.g.*, cGANs generate a cat image based on a "cat" query. However, cGAN-based synthesizers can be stolen via adversaries' queries, *i.e.*, model thieves. The prevailing adversarial assumption is that thieves act independently: they query the deployed cGAN (*i.e.*, the victim), and train a stolen cGAN using the images obtained from the victim. A popular anti-theft defense consists in throttling down the numbers of queries from any given user. We consider a more realistic adversarial scenario: model thieves collude to query the victim, and then train the stolen cGAN. CLUES is a new collusive model stealing framework, enabling thieves to bypass throttle-based defenses and steal cGANs more efficiently than through individual efforts. Thieves collect queried images, and train a stolen cGAN in a federated manner. We evaluate CLUES on three images datasets, *e.g.*, MNIST, FashionMNIST and CelebA. We experimentally show the scalability of the proposed attack strategies against the number of thieves and the queried images, the impact of a classical noise-based defense, a passive watermarking defense and a JPEG-based countermeasure. Our evaluation shows that such collusive stealing strategy gets close to 4 units of Frechet Inception Distance from a victim model. Our code is readily available to the research community: https://zenodo.org/records/10224340.**

I. Introduction

Data synthesizers are increasingly offered as a service on the web to augment the data quantity and diversity. In a nutshell, users issue queries to such a service (*i.e.*, one generating animal images) and receive the corresponding images back. We list several popular examples in Table I. At their core, such synthesizers exploit generative models and conditional generative adversarial networks (cGANs) [47], used to synthesize images [19], [62], music [18], generic [36] as well as specialized time series [61], tabular data [64], [71], videos [15], [46] and more. cGANs consists of two networks, the generator G^T and discriminator D^T, both of which are iteratively and jointly trained (see Fig. 1). Only the generator is used in the deployed service. Training cGANs is extremely challenging, requiring a large number of input samples (*e.g.*, real images) and computation power to tune the hyperparameters. Consequently, deployers face the challenge to keep key model information, *e.g.*, model architecture and weights, confidential and inaccessible by the users (*i.e.*, black-box model deployment [54]), as well as shielded from compromised nodes serving the model to prevent unauthorized access, misuse, and privacy compromises [56].

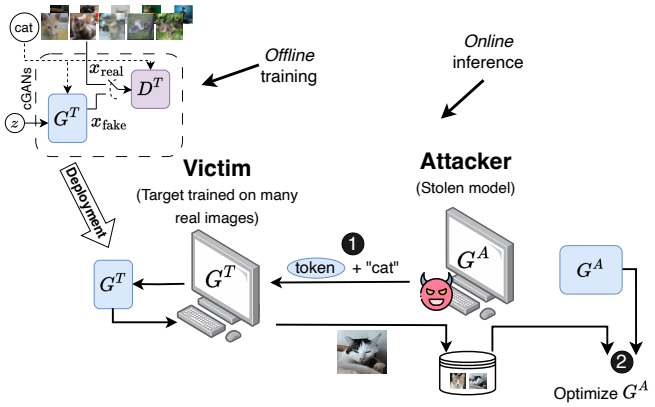

Fig. 1: Two-step model stealing: ❶ query victim's generator model G^T via a condition prompt (plus an identification token needed to access the service) and collect the synthesized outputs, ❷ optimize stolen generator model G^A to produce the same result.

Despite the efforts of hiding the machine learning models behind such services, there is the risk that adversarial parties steal the models through querying the services. Several stealing attacks [35], [66] have already proven effective on classification models. A recent study [32] has shown a stealing attack effective on deployed GANs generators. In their case, the thief first sends a request and receives the corresponding synthesized image from the victim's GAN generator with no regards for the attacker's topology (this aspect is later discussed further in this section). We extend the attack to incorporate GANs/cGANs (see Fig. 1). In our example, the attacker prompts the victim with a "cat" class to obtain a picture of a cat generated by G^T. Afterwards, the query inputs (*i.e.*, condition prompts) and outputs (*i.e.*, generated images) are used by the thief as inputs to train a generator G^A such that, for a set of classes, the quality of its synthesized images is as close as possible to the deployed victim cGANs generator's.

Generally speaking, the strength of such stealing attacks highly depends on the number of adversarial queries, *i.e.*, high number of synthesized images, for better quality of the stolen model. A typical defense strategy to minimize the risk

Authors' copy of Simon Queyrut, Valerio Schiavoni, Lydia Y. Chen, Pascal Felber and Robert Birke (2024). CLUES: Collusive Theft of Conditional Generative Adversarial Networks. In proceedings of the 43rd International Symposium on Reliable Distributed Systems (SRDS 2024). In press.

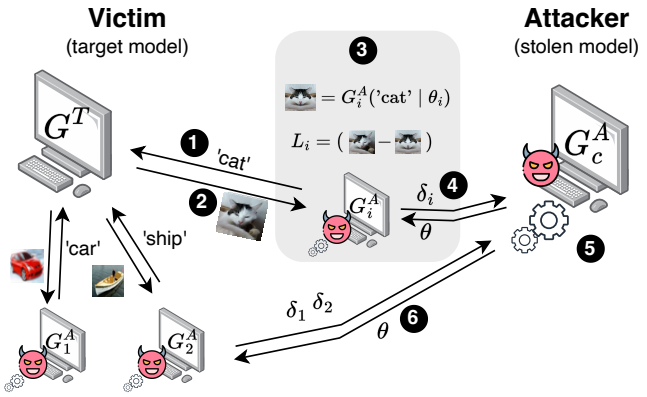

Fig. 2: Colluding stealing strategy to evade query throttling: colluding agents query the victim's target generator G^T ❶ which answers with synthesized images ❷. Each agent i computes δ_i, the gradient of the loss between the victim's and local model's generated samples ❸ and sends it to a central server ❹; it averages all received gradients to update the central model $G_c^A(\cdot|\theta)$ ❺. Finally, the updated weights θ are sent back to all agents as starting local model for the next federated learning round ❻.

of model theft is thus to limit the queries per user. This is widely adapted by cloud service providers including those hosting generative AI services [12], [14], [25] and typically implemented by attaching a client-identifier token to each query so that providers can throttle abuses of their endpoints.

Despite the threats of such attacks, the prevailing assumptions in state-of-the-art approaches [32] are twofold: *(i)* malicious users have large communication and computation capacities to retrieve high number of samples from the victim and train the stolen model; and *(ii)* they act independently without any collusion.

In reality, adversarial parties are largely heterogeneous and, henceforth, differ in their computing capacities, typically orders of magnitude below the machines serving the victim models (*i.e.*, the model owners). To effectively and efficiently steal the victim model while at the same time overcome throttling defensive mechanisms, model thieves can resort instead to *collusive stealing*, a new model theft tactic that we contribute in this work. In collusive stealing, several agents (*i.e.*, the model thieves) cooperate during both the query phase as well as the training of the stolen generator. To the best of our knowledge, existing literature has largely ignored collusive stealing attacks. However, as we show next, these can stealthily undermine the defenses of services leveraging trained (c)GANs.

In this paper, we investigate collusive stealing attacks on cGANs, as they are among the most prevailing image generative models used. The victim generator of cGANs is accessed through public API queries. Users specify the class labels (*e.g.*, "cat") and receives in return the queried images. The victim server imposes a limit on the number of queries per user. We consider that model thieves have limited computation and communication capacities.

We contribute CLUES, the first collusive stealing strategy for deployed cGANs, where multiple thieves jointly train a stolen cGANs via queried images. Further, CLUES contributes an optimization algorithm that minimizes the *L1 distance* (*i.e.*, the total absolute pixel-to-pixel difference for each channel) between images generated by the victim and stolen cGANs. Considering the heterogeneous nature of model thieves and aiming for high system scalabiltity, we propose a colluding strategy based on model federation (see Fig. 2). Each thief first collects the queried images. Then, it trains a stolen generator with other thieves in a federated manner [45]. Model federation parallelizes querying the victim according to imposed limits and also the stolen generator's training effort.

TABLE I: Observed per-free-account rate limits of some popular data synthesizers (image generation). Quantities marked with an asterisk (*) denote daily limits and those marked with a dagger (†) were the obtained quantities after we voluntarily stopped querying.

Service provider	Rate limit	Service provider	Rate limit
Replicate [9]	100	Microsoft Designer [8]	400
DatumBox [4]	1000*	ImageFX [7]	1200*
AI/ML API [3]	1307	Dream by Wombo [5]	2109†
Eden [6]	3333	thisxdoesnotexist [10]	5000†

As victims can hardly discern thieves from normal users, collusive stealing easily circumvents defenses based on query limits. To lower the risk of model theft, we propose a strategy that perturbs the generated image with defensive noise. We leverage the fast gradient sign method FGSM [27] to craft noise that does not visually impair the output images but pushes the stolen model optimization towards maximum loss, disrupting query-based model training.

Our evaluation across three popular datasets (*e.g.*, MNIST, FashionMNIST and CelebA) shows that CLUES can successfully steal a victim model, and that, even for a constrained number of queries, the collusion of agents steadily outperformed agents working alone. We explore the mitigating effects of a noise-based defense and a passive watermarking defense against our proposed scheme.

Our contributions can be summarized as follows:

- CLUES, the first collusive stealing strategy, demonstrating the realistic and stealthy threat against deployed image generating services (see §III);
- a new optimization algorithm to train a stolen generator by minimizing the L1 distances of images between the victim and stolen models (see §III-C);
- a colluding strategy to parallelize both model querying and training of a stolen generator in a federated manner amongst the thieves (see §III-D);
- a noise-based defense strategy, leveraging gradient inversion to disrupt the training of a stolen generator (see §V);
- an adaptive stealing which counters the defense (see §V-B).

Roadmap. In §II we explain the mechanism behind conditional GAN services. We detail the architecture of CLUES in §III. §IV presents our experimental results. §V describes defenses and counter-attacks (adaptive stealing) against CLUES and the corresponding costs. We survey related work in §VI, before concluding and presenting future work in §VII.

II. Background on GAI services

In this section, we first describe how generative models are served through API. Then we detail the internals of cGANs.

Generative AI Services. Modern generative AI services allow to synthesize images, music, tables and other types of data based on some user friendly prompt, *e.g.*, a class label, plus some randomness, *e.g.*, a noise vector. The power of the models behind such services is deeply connected to their size and size of training data [33]. Consequently, *(i)* large trained generators, *i.e.*, model architecture and weights, are an asset to be protected, and *(ii)* inference at scale requires abundant resources. Deploying such models in the Cloud can cater to both. In this work, we consider the classic scenario of a cloud service provider exposing an extensively trained generative model via a freely accessible or paid API service (*e.g.*, ML-as-a-Service [1], [2], [11]). In particular we focus on black-box model serving which, in contrast to white-box model serving, hides any model details (model layers, functions or weights) and inference activity (gradients, activations or any numerical value/measure created during inference) from API invocations. Only the final synthesized data is returned as result.

In such a setup, a malicious end-user aims to train a stolen model which behaves indistinguishably to, *i.e.*, synthesizes data of same quality as, the victim model. The malicious user could be endowed with a generator of similar architecture, constructed through metamodel techniques [51] or obtained by a wild guess using a side channel (if the target disclosed the genus of their model on their website, for example). He could also have a model of a different architecture yet fulfilling the same tasks. He repeatedly queries the deployed model and iteratively updates his local model to match the output of the service (see Fig. 1 for a description of the workflow). If a sufficient number of queries is completed, the malicious client can successfully train a generator with equivalent performance to the model being queried, effectively stealing the model.

To diminish the effectiveness of the efforts of malicious users, service providers use query throttling as common defense mechanism. As a mere illustrative example, we gathered rate limits through empirical observation while requesting the free plans of various online generative APIs. We solicited endpoints for four hours each and recorded the number of samples obtained until the generation service ceased (we chose to halt [10] at 5000 to align with the scope of our evaluation). The results are presented in Table I. However, in contexts involving individual-restricted queries, *e.g.*, only input-output interactions are permitted, malicious users can parallelize data acquisition by creating multiple (trial) accounts for the service. CLUES is relevant in such scenarios.

We assume that CLUES operates under the most stringent assumptions regarding attackers. In particular, the victim model is enclosed within a black box that exposes a generative API hiding any model internals. As a result, the victim holds a target generative model of good performance whose weights and architecture cannot be attained through any other method or side channel. This incentivizes the attacker to allocate resources and employ a suitable strategy to match the performance of the target model by training its own model.

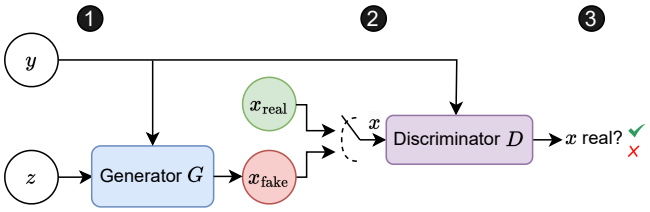

Fig. 3: Basic workflow for cGANs training: ❶ G is fed a noise vector z and a condition y to produce a fake image x_{fake}. Next, fake x_{fake} and real x_{real} samples are randomly fed to D ❷ alongside the condition. During training, D should recognize real from fake data, so it is rewarded when it correctly classifies x as real or fake ❸.

While our study primarily deals with conditional GANs, we emphasize that CLUES is not limited to vanilla GANs or other conditional models. Instead, our objective is to formulate a strategy that proficiently extracts the performance of a target generative model by employing a combination of recurrent queries and orchestrated collaboration among thieving agents.

Conditional Generative Adversarial Networks (cGANs) are a type of generative model used to produce synthetic samples, typically images, based on a class label y and a noise vector z of dimension n_z. The choice of n_z is critical for both training and inference, as it affects the complexity of the mapping between the latent (noise) space and images. Noise dimension influences the balance between exploration and exploitation: adjusting the noise level allows control over the generator's exploration of potential outputs.

cGANs comprise two contrasting models, namely a generator G and a discriminator D. The goal of G is to produce realistic image-label pairs able to fool D. The goal of D is to discern fake from real example-label pairs. It learns to accept correctly matched pairs while rejecting mismatched pairs and fake examples (see Fig. 3). G is a differentiable function, implemented as a deep neural network (DNN). It is parameterized by θ_g designed to learn how to synthesize realistic examples, $x_{\text{fake}} = G(z|y;\theta_g)$ for each class label y inside a training dataset of images $\boldsymbol{x}_{\text{real}}$ of assumed underlying conditional distribution p_{real}. For brevity, we write $G(z|y)$. To learn a conditional distribution p_g of the generator G over $\boldsymbol{x}_{\text{real}}$, we sample z from a prior distribution p_z, such as Gaussian distribution or uniform distribution. During training, the generated (fake) images x_{fake}, along with real images $\boldsymbol{x}_{\text{real}}$ and their labels, are randomly fed to a discriminator D, another DNN which assesses the likelihood of each image being either fake or real. Similarly, D is parameterized by θ_d and we abbreviate it as $D(x|y;\theta_d) = D(x|y)$. The output of D is a single scalar value for each input image, indicating the probability of whether the input is real or fake (*i.e.*, the probability that x_{fake} is drawn from p_{real} rather than p_g). The output is a continuous value, often in the range $[0, 1]$, where values closer to 1 represent high confidence in the input being real, and values closer to 0 represent high confidence in the

input being fake. Within the generator, the prior input noise z and the conditional variable y are amalgamated into a shared hidden representation. As such, for the sake of conciseness, we often note $G(z|y) = G(y)$. Once trained, the discriminator is dropped and only the generator is deployed as a service. The intent of the malicious user is to clone the victim's generator rather than using the queried data to train a new generator.

III. Architecture of CluеS

The goal of CluеS is to enable multiple thieves to collude and replicate a target conditional generative model and achieve high-quality synthesis. During the attack, they collectively train a generator to mimic the target generator.

A. System and Adversary model

The victim (or target) model G^T is a generative model deployed and hosted remotely, *e.g.*, accessed via a wide-area network and exposed through an API. It is considered to be inside a black box: given input (query), returns output (generated sample). Thieves collude to reproduce G^T. Since it is assumed that the potentially malicious client has control over the random seed to ensure generation consistency, we allow the clients to query the victim with their own noise. We consider i colluding agents (participating nodes are not byzantine with respect to one another), which repeatedly query the generative API under a constraint on the number of queries, typically set by the victim. Agents can exchange updates of the shared model with a central server but cannot interact with one another. Colluding agents share a common model G_c^A whose authoritative version is held by a central server and regularly broadcast to the agents for inference. The central (or aggregating) server interacts with all thieves individually. Each time agents receive an answer to their query, they may use this sample to compute the gradients to send to the central server.

B. Architecture of CluеS

Each thief node must have access to the desired service API and establish a connection with the central server. They then send enough information upwards about their device and API access such that the central server assigns the agents a task they can handle computationally, memory-efficiently, and budget-wise. The agreement between the agents and the central server on communication protocols, as well as the overall federation approach, is a design specific to each use case, with many documented instances [16]. This step is nonetheless crucial as it contributes to establishing and orchestrating the attacking strategy to be followed by the central server. It enables the central server to become aware of available resources and ensures adherence to the limitations of each device's memory capacity and allocated API query budget. This strategy includes but is not limited to: fraction of participating devices, update round frequency, convergence criteria (*i.e.*, when to stop the theft), number of local epochs to be carried on the workers, etc.

Agents interact with the central server but not with one another because they do not know each other and have an interest in remaining anonymous from one another; *e.g.*, in the case of a botnet, the primary communication channels are between the bots and the central server. Establishing direct communication between individual bots presents additional challenges (*e.g.*, synchronization protocols, topology optimization) and brings potential benefits (*e.g.*, improved querying efficiency, sharing computing resources), which are left for future work.

C. Optimization problem

We formally model the stealing process of CluеS where N_a agent sends $N_{q/a}$ queries each consisting of a label-noise pair (y, z) to the target victim generator G^T. In turn, it sends the responses $x = G^T(z|y)$ back to the agents, thus constituting a local dataset. Given an underlying conditional distribution p_{real} which is explicitly or implicitly determined by G^T (and the strategy of CluеS try to reproduce), *i.e.*, a sample x collected from G^T given y is generated by p_{real}. With these assumptions, G_c^A aims at minimizing the risk:

$$\min_{\theta_c} R(G) = \mathop{\mathbb{E}}_{(x,y)\sim p_{real}} [\ell(x, G_c^A(z|y;\theta_c))].$$

When dealing with images, $\ell = \|\cdot\|_1$ is the pixel-by-pixel L1 distance. In the case of CluеS, G_c^A iteratively optimizes its parameters θ_c by averaging the gradients received from each federated agent (indexed by i) and computed as $\nabla_{\theta_i} \left\|x - G_i^A(z|y;\theta_i)\right\|_1 = \nabla_{\theta_i} \left\|G^T(z|y) - G_i^A(z|y;\theta_i)\right\|_1$.

We note that this objective function enforces the stolen generator G_c^A to imitate the output of the victim generator in an exact fashion. Another alternative to make use of queried images is to train cGAN from scratch, using both generator and discriminator networks. Such an alternative has several disadvantages. Firstly, the cGAN training intends to capture the underlying distribution of the data sets and learns to generate images that are similar but not identical to the real images. Secondly, the GAN training is difficult to train and suffer from the instable convergence and mode collapse [27].

D. Collusive Model Stealing

Due to aggregation delay at the central server and communication latency, thieves' G_c^A versions differ from the central node's. We denote each thief i's model as G_i^A, parameterized by θ_i.

Overview. Algorithm 1 and 2 describe the server and agent-side of collusive stealing process. The aim is to minimize the empirical risk of an architecture G_c^A with a constraint on the number of queries $N_{q/a}$ at the disposal of each of distributed N_a agent. Broadly speaking, it leverages FedSGD [45] where in rounds each agent queries the target G^T with label and noise, computes the loss between the reply and previous round model output in order to perform gradient computation, and sends the gradients to the central server for updates to the stolen model. FedSGD guarantees good model convergence at the cost of high communication costs stemming from the frequent exchanges of gradients and model weights. However, as an alternative to transmitting voluminous datasets to a centralized server, it solely conveys model updates, thereby yielding substantial reductions in both bandwidth consumption

Algorithm 1: CLUES server

Input: N_a, $G_c^A(\cdot|\theta_c)$, optimizer
Variables: $\{\delta_i\}_{1\leq i\leq N_a}$, Δ
Result: $\theta_c^* = \text{argmin}_{\theta_c}\ R(G_c^A)$

1 **foreach** round **do**
2 **foreach** $i \in [\![1, N_a]\!]$ **do**
3 sendAgent(i, $G_c^A(\cdot|\theta_c)$)
4 **foreach** $i \in [\![1, N_a]\!]$ **do**
5 $\delta_i \leftarrow$ getUpdate(i)
6 $\Delta \leftarrow \frac{1}{N_a}\sum_{i=1}^{N_a}\delta_i$
7 $\theta_c \leftarrow$ optimizer(θ_c, Δ)
8 **return** θ_c

Algorithm 2: CLUES i-th agent

Input: $N_{q/a}$, p_y, p_z, B
Variables: $(G_i^A, \delta_i, x, z, y, \mathcal{D}_i)$

1 **while** *True* **do**
2 $G_i^A \leftarrow$ getModelFromServer()
3 $\delta_i = 0$
4 $b \leftarrow \min(B, N_{q/a})$
5 **foreach** $j \in [\![1, B]\!]$ **do**
6 **if** $j \leq b$ **then**
7 $(y, z) \leftarrow$ (rand $y \sim p_y$, rand $z \sim p_z$)
8 $x \leftarrow$ queryTarget(z, y)
9 $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup (x, y, z)$
10 **else**
11 $(x, y, z) \leftarrow$ sample($\mathcal{D}_i$)
12 $\delta_i \leftarrow \delta_i + \nabla_{\theta_i}\left\|x - G_i^A(z|y_j)\right\|$
13 $N_{q/a} \leftarrow N_{q/a} - b$
14 sendToServer(δ_i)

and server load [45]. Additionally, past queries are saved in a local datastore $\mathcal{D}_i$ so they can be reused over multiple rounds and increase the data efficiency in terms of API requests.

We emphasize that the strategy outlined in Algorithm 1 and 2 is entirely data-type agnostic. No assumptions are made regarding the nature of the generations, which encompass the full range of data types supported by conditional generators (which, in turn, are not confined to cGANs).

Stealing parameters. As stated previously, it is the task of the initiator of the collusive stealing effort to set the protocol to be followed for the theft. System, performance and security measures are involved in this choice, but for the general and formal outline of CLUES, these stealing strategy parameters are listed as inputs of Algorithm 1 and 2 .

To send queries we need to sample prompts (or labels) y and noises z_i from the prompts and noise distributions, namely p_y and p_z. Choosing p_y should be done as to mitigate class imbalance in the queried images. Processing rounds is organized in batches of queries of size B. The batch size B plays a role in achieving the desired tradeoff in convergence and communication overhead, influencing the overall efficiency of the theft. Moreover, a batch sufficiently small to be held in memory can speed computations. Model weights and biases are drawn from a uniform or Gaussian distribution: CLUES in addition supports also more sophisticated initialization methods [29]. CLUES is independent from the choice of the optimizer which encompasses also the choice of the learning rate and various tunable parameters of the optimizing function.

Starting a new round. The central server initiates each new round by sending G_c^A (Alg. 1-line 2-3) and then waits for new gradients from each agent (Alg. 1-line 4-5).[1]

Querying the victim model. Each agent receives a copy of G_c^A (Alg. 2-line 2). Based on the remaining query budget, the agent decides how many b queries to send and process out of B (Alg. 2-line 4). For each query, the agent samples noise z and prompt y vectors (*i.e.*, the desired class of the generated sample) from p_y and p_z (Alg. 2-line 7) and queries the victim to obtain a reference sample x (Alg. 2-line 8). x together with y and z are stored in a local datastore $\mathcal{D}_i$ (Alg. 2-line 9). If the query budget is depleted, the next sample, *i.e.*, $j > b$, is randomly drawn from $\mathcal{D}_i$ instead (Alg. 2-line 11).

Computing per batch gradient. The agent uses the sample x to compute the loss with respect to a locally generated counterpart based on (z, y) and accumulates the derived gradient locally for the whole batch B. (Alg. 2-line 12). Once a whole batch has been processed, the agent updates the query budget (Alg. 2-line 13) and sends back to the central server the final gradient (Alg. 2-line 14).

Server aggregating clients' gradients. Once the server has received all gradients, they are averaged as Δ (Alg. 1-line 6). In case of failure from one of the participants (*e.g.*, an agent sends corrupted gradients or is no longer trusted by the central server), such gradients can be discarded, and the weighting value N_a is decremented by one. Δ in turn is passed to the optimizer to update the model parameters θ_c (Alg. 1-line 7). After that iteratively a new round starts repeatingly optimizing the central model (G_c^A) to gradually approximate the performance of the target model (G^T).

IV. EVALUATION

We present here our extensive performance evaluation using both simulations and a full-fledged prototype. We postpone to §V an evaluation and discussion of defenses and counter-defenses. Our experiments intend to answer the following questions: **(Q1)** Can CLUES steal the generator networks of target cGAN model? **(Q2)** What are the trade-offs in FID performance of CLUES's collusive stealing?

A. Implementation details

We leverage PyTorch v2.1 and Flask v3.0 to expose a REST-based API in real-world deployments. The code, models, and detailed instructions to reproduce all our experiments are available at https://zenodo.org/records/10224340. The architecture and the implementation of the layers inside the various mentioned DNN models levereged by CLUES are best described on the official website of PyTorch [13]. Experiments in this section were run on a Tesla V100S 32GB.

[1] In simulation the *get_update()* directly calls the agent side computations.

TABLE II: CLUES's side experimental setup on three datasets.

G_c^A **Training hyperparameters**	
MNIST/FashionMNIST	CelebA
#rounds: 900 Batch size: 128	#rounds: 600 Batch size: 64
Loss: L1 n_z: 100	Loss: L1 n_z: 100
Learning rate: 0.922/0.0012	Learning rate: 0.0032
Optimizer: Adam	Optimizer: Adam
β_1, β_2: 0.5, 0.999	β_1, β_2: 0.5, 0.999
Filters: 8 image size: 32	Filters: 124 image size: 64
p_z, p_y: uniform	p_z, p_y: uniform
Experiment hyperparameters	
MNIST/FashionMNIST	CelebA
Num. runs per $(N_{q/a}, N_a)$: 10 (except $N_a = 50$: 5 runs)	Num. runs per $(N_{q/a}, N_a)$: 5 (except $N_a = 1$: 10 runs)
Loss: L1	Loss: L1

TABLE III: Best FID (mean±st.dev.) by the stolen model over 900 rounds for several numbers of thief-agents N_a and number of queries per agent $N_{q/a}$.

	MNIST		
	Baseline; Victim FID $= 40.76$		
$N_{q/a}$	1	10	50
100	184.02±16.48	150.63±10.74	172.20±15.95
250	150.54±9.72	126.25±7.35	159.10±5.63
500	133.67±9.24	121.45±7.42	120.32±2.38
1000	100.1±7.81	96.62±5.82	89.77±4.01
1500	73.27±8.33	72.11±6.21	77.52±3.08
3000	53.73±4.40	44.53±4.26	55.61±4.15
	FashionMNIST		
	Baseline; Victim FID $= 36.71$		
$N_{q/a}$	1	10	50
100	96.17±5.50	97.65±6.76	119.02±3.21
250	92.99±4.17	89.64±4.02	92.33±6.81
500	86.33±3.98	83.63±4.34	83.16±3.15
1000	77.71±5.06	73.85±6.10	73.66±4.72
1500	65.13±6.12	67.7±5.31	64.88±3.33
3000	61.15±1.92	58.43±2.38	62.18±2.90
	CelebA		
	Baseline; Victim FID $= 28.40$		
$N_{q/a}$	1	10	50
500	104.98±6.48	100.99±5.74	100.41±5.65
1000	99.02±2.42	98.25±3.11	98.4±3.08
2000	96.79±4.01	95.9±5.26	95.69±3.38
3000	94.34±3.81	93.17±3.64	95.17±3.23
4000	91.35±5.33	91.41±3.21	91.60±3.18
5000	90.8±4.18	88.8±5.02	91.78±4.22

B. Evaluation setup

We consider several scenarios with varying numbers of federated agents (up to 50), subject to constraints on $N_{q/a}$ across three datasets, namely MNIST [24] (10 classes), FashionMNIST [63] (10 classes) and CelebA [43] (2 classes), given the predominance of image generation's application domain for cGANs. To reflect real-world conditions imposing throttling limitations (§III), we limit the dataset collection process to conform to diverse querying thresholds $N_{q/a}$ (from 100 up to 5000 samples from different classes as to meet reasonable values we observed and listed in Table I). We vary the number of thieves N_a to observe the impact on theft performance, *i.e.*, on the quality of the image output from the stolen model G_c^A held by the central (aggregating) node of the federated attack. The outcomes are outlined in Table III and described in the remaining of this section.

Victim model. The victim Generator G^T and Discriminator D^T are convolutional neural networks following the implementation of [47]. We encourage readers to refer to the supplementary material refered to in §IV-A for a more in-depth understanding of the generator's and discriminator's architecture and training parameters. Both the Generator and Discriminator are trained using Binary Cross-Entropy Loss and the Adam optimizer with the same learning rates. For each of the three datasets, MNIST (6e4 samples), FashionMNIST (6e4 samples) and CelebA (over 2e5 samples), we train a victim model with hyperparameters detailed in the supplementary material. Training (Fig. 1) happens before launching the CLUES stealing attack. For each of the three datasets, D^T is dropped and G^T is fixed, acting as a query-in-answer-out API all throughout the theft.

Stolen model G_c^A. The attacker ignores the exact victim's generative architecture. G^T is anticipated to be very sophisticated. This expectation is grounded in the knowledge that the developers responsible for the API constitute dedicated teams with specialized expertise, resources, and extensive datasets. The utilization of a larger model is inferred from the presumption that a substantial dataset necessitates a model of greater size to be efficiently exploited as larger models often generalize better [33]. As such, we implement this scenario by endowing the thief with a cGAN architecture with a smaller dimensionality of its feature space. Specifically, it features fewer output filters for each of the convolutional layers inside G_c^A . Table II details the chosen hyperparameters and all used $(N_{q/a}, N_a)$ pairs (including number of runs per pair). Notice the number of rounds is large enough to allow each agent to go through several epochs of their $N_{q/a}$, *e.g.*, 7 times for CelebA as $600 > 7 \cdot (5000/64)$. For the pairs of each dataset, to mitigate bad sampling of the noise and labels (*i.e.*, p_z and p_y), we run CLUES several times against G^T. Rather than considering the G_c^A of the last round as the best model to return from the theft (Alg. 1-line 8), we systematically take the best Frechet Inception Distance (FID) reached by G_c^A throughout the rounds of training.

Frechet Inception Distance (FID). In this study, we use the Frechet Inception Distance [50] (or score) to assess the performance of the stealing process. In a nutshell, FID evaluates image quality and diversity, indicating that the Generator should produce images spanning various recognizable classes. FID score estimates divergence, extracting features using an Inception v3 network [60], and modeling data distribution (in our case, 2048 samples) using a Gaussian distribution with mean and covariance. We compute FID every 100 rounds between a batch of 2048 samples generated by G_c^A when fed uniformly sampled y labels and a fixed batch of 2048 samples drawn from the training set of the victim (*i.e.*, real images). For reference, we also compute the FID of 2048 samples

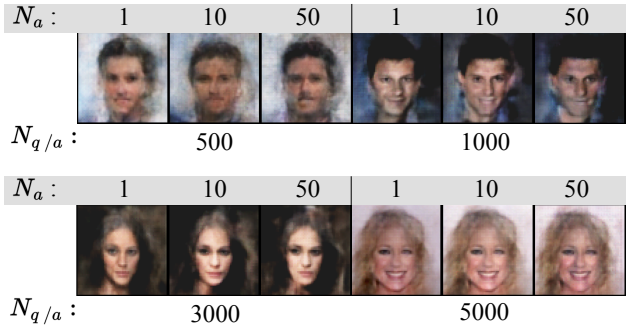

Fig. 4: Outputs from stolen model against a CelebA-trained victim.

generated by the victim G^T when its training is over for each dataset (also shown in Table III). Note that the optimization problem from §III-C is equivalent to the minimization of the difference between *(i)* the FID of G^T's output distribution with the distribution of real images and *(ii)* the FID of G_c^A's output distribution with the distribution of real images. In summary: lower FID indicates that the distribution of features in the generated images is closer to the distribution of features in real images *i.e.*, lower FIDs indicate more successful stealing.

C. Can CLUES steal a cGAN with distributed agents?

Table III shows the best FID obtained on average (next to the corresponding standard deviation values) over several runs. The exact number of run is reported in Table II, for each $(N_{q/a}, N_a)$ pair for all the three datasets. For any N_a, as the quantity of training data $N_{q/a}$ increases, the FID consistently lowers, indicating a learning pattern. With many queries per thief, for MNIST we reach a FID of 44.5, very close to the FID of the victim, *i.e.*, 40.76. Due to more complex feature distributions in FashionMNIST and CelebA, performance of the theft decreases because $N_{q/a}$ remains negligible when compared to the amount of data used for training the target. The best G_c^A obtained for FashionMNIST came close to target by 12 units of FID and 60 units of FID in the case of CelebA.

Figures 4-7 display some samples generated by the best G_c^A over the rounds (in terms of FID) and a sample generated by G^T. Visual likeness of attacker-generated samples with victim sample improves as $N_{q/a}$ increases and confirms the performance of the stolen cGAN.

A 1 : *Even with a small number of queries ($< 3\times10^3$),* CLUES *can train a generator G_c^A of nearly identical performance of the target G^T (trained on over 6×10^4 samples in the case of MNIST). Overall, FID values display improvement in performances as $N_{q/a}$ increases, confirming the effectiveness of the stolen generator.*

D. Trade-offs of CLUES's distributed stealing

Table III shows that federated stealing with $N_a = 10$ agents consistently outperforms the scenarios of a lonely thief ($N_a =$ 1) sending gradients. This is the proof of the generalization capabilities of the generator, due to the larger variety of samples collected (*i.e.*, victim is queried $10\times$), a well-known benefit of federated learning [39]. Notice that experiments for which the $N_{q/a}{\cdot}N_a$ product is equal systematically favor low N_a because it constrains the central model to a training behavior that resembles mini-batch gradient descent (GD), *i.e.*, updates after each batch, whereas a high N_a means a similar model (*i.e.*, not updated) computes gradients over ever smaller batches, and thus converges more slowly throughout the epochs of which there are fixed numbers (§IV-B). On average, $N_a = 10$ improves by 13.96 points in FID over $N_a = 1$ in MNIST, 2.78 in FashionMNIST and 1.46 in CelebA. However, in other cases, we did observe a deterioration of performance compared to the $N_a = 1$ and $N_a = 10$ counterparts, *e.g.*, for FashionMNIST at $N_{q/a} = 100$, $N_a = 50$ is worse than the less-agents case by over 20 units of FID. We explain this by the tradeoff between generalization capabilities endowed by a much larger pool of samples and speed of convergence of the stolen model. The large dataset improves the overall learning process of G_c^A, but it still fails to learn the noise-to-image mapping within the given rounds using an optimizer that is fit for faster convergence. On average, $N_a = 50$ offered an improvement of only 8.37 points in FID over $N_a = 1$ in MNIST, 1.42 in FashionMNIST and 0.71 in CelebA.

A 2 : *Distributed stealing with $N_a > 1$ improved the quality of the stolen generator by up to 14 points in FID compared to the $N_a = 1$ case. The improvement brought by involving several participants tend to decrease when N_a is too high.*

TABLE IV: Mean best FID and standard deviation achieved by the stolen model over 900 rounds across a range of the number of thief-agents N_a and number of queries per agent $N_{q/a}$ for the case where the victim applies an FGSM mask on its output images (lower values favor the thief).

	MNIST		
	Baseline + FGSM ($\epsilon = 0.1$); Victim FID $= 43.46$		
$N_{q/a}$	1	10	50
100	181.97±9.42	166.44±3.81	174.30±7.44
250	164.69±4.69	155.08±4.28	162.44±2.94
500	143.51±5.16	145.81±3.69	150.64±4.89
1000	122.12±2.22	117.36±2.85	132.36±3.45
1500	102.86±3.63	98.82±4.95	111.03±4.27
3000	68.32±6.24	64.19±2.94	72.54±4.01

	FashionMNIST		
	Baseline + FGSM ($\epsilon = 0.03$); Victim FID $= 41.66$		
$N_{q/a}$	1	10	50
100	167.48±8.70	175.92±5.26	179.79±8.17
250	159.33±5.22	165.28±6.90	155.52±7.21
500	135.2±3.97	129.54±6.65	136.02±10.81
1000	116.41±3.89	107.24±7.25	118.34±6.51
1500	97.08±4.12	95.80±3.84	98.82±7.05
3000	91.52±3.04	88.15±5.12	93.47±6.88

TABLE V: Mean best FID and standard deviation achieved by the stolen model over 900 rounds across a range of the number of thief-agents N_a and number of queries per agent $N_{q/a}$ for the case where the victim applies an FGSM mask to its outputs as a defense but the agents run the queried images through JPEG before computing their gradients (lower values favor the thief).

	MNIST		
	Baseline + FGSM + JPEG; Victim FID = 43.46		
$N_{q/a}$	1	10	50
100	188.92±18.32	158.73±11.46	171.62±14.03
250	158.2 ±10.91	148.09±9.34	166.34±7.22
500	141.02±10.20	132.94±10.65	134.57±5.56

	FashionMNIST		
	Baseline + FGSM + JPEG; Victim FID = 41.66		
$N_{q/a}$	1	10	50
100	97.01±4.90	102.31±6.36	123.98±3.17
250	94.72±6.47	91.44±5.89	99.18±10.48
500	90.53±3.33	86.31±5.29	85.09±3.41

V. Defense and Adaptive Stealing

In this section we intend to answer the following questions: **(Q3)** Can victims effectively defend against CLUES using adversarial perturbations? **(Q4)** Can victims effectively defend against CLUES using watermarking?

Watermark is widely adopted to protect the copyright of documents and images, but its effectiveness against functional stealing is questionable. We explore victim fingerprinting in §V-C. Recent gradient-based watermarking [38] for classification models hinders thieves from learning input-output mappings; there, adversarial perturbations in synthetic images impede proper learning of victim diffusion models.

The Fast Gradient Sign Method (FGSM) [27] is an attack that perturbs a data sample x to craft adversarial examples (inputs to a machine learning model intentionally designed to cause the model to misclassify the input data) by nudging x towards a higher loss for a model through gradient ascent with an magnitude factor ϵ. To validate FGSM defensive capacities, we let victim model apply a FGSM mask (we chose $\epsilon = 0.01$ as the generated mask barely affects the victim yet heavily penalizes G_c^A) on its generated outputs before returning the image to the querying node. We deploy the same experiments as earlier (§IV-D) for MNIST and FashionMNIST, but now the victim uses the LeNet-5 [41] in its ReLU variant to compute the defensive overlay, layed over each of its responses. Table IV presents our results. We observe the following: firstly, the FGSM mask significantly reduces the performance of the model that attempts to copy it. For MNIST, the resulting FID of the attacker was increased by 19.61 on average. For FashionMNIST, it increases by 48 points in FID. The standard deviation for the runs also increased (+2.89 for MNIST and +1.71 for FashionMNIST) indicating a less reliable training process. Secondly, this defense does not *a priori* completely counter the distributed stealing, since FID scores continue to improve as $N_{q/a}$ increases: this suggests that FGSM alone is not sufficient and additional mitigation techniques are beneficial. All in all, the victim was able to hinder the theft process using the FSGM mechanism, whilst sacrificing relatively little performance: the FID of G^T's output distribution was increased by only 2.7 for MNIST and 4.95 for FashionMNIST.

TABLE VI: Setup for the experimental test-bed.

	CPU (Cores)	**Memory**	**GPU**
Agent	AMD EPYC 7302P (32)	32 GB	-
Server	Intel Xeon E3-1270 v6 (8)	64 GB	-
Victim	Intel Xeon Skylake IBRS (16)	258 GB	✓

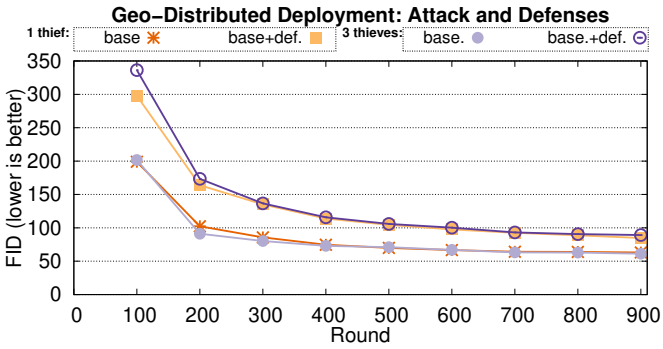

Fig. 5: Geo-distributed attack using CLUES and FSGM defense over FashionMNIST. Results with 1 or 3 thieves. FID score achieved by the stolen cGAN at each 100 rounds averaged over 5 runs.

A. Geo-distributed deployment: attack and defense

We validate how the distributed stealing and the FGSM protection behaves by mean of a geo-distributed deployment. We deploy 1 victim server and 3 thieves, connected over a WAN network. Table VI reports the hardware characteristcs of these nodes. Agents run off-the-shelf hardware. The round-trip network latency between the agents and the server is 13.3 ms. The average response time for one batch of queries consisting of 128 label-noise pairs is 0.42 s and 0.43 s, with and without FSGM (indicating a relatively neglectable overhead in applying this defense). Fig. 5 depicts the evolution of the FID during 900 rounds. We observe how the 3-thieves distributed attack leads to a better stolen cGAN (*i.e.*, lower FID), and that the FSGM defense effectively leads to overall poorer results (*i.e.*, higher FID). Note that the stealing process in this setting occurs within minutes, which is several orders of magnitude shorter than the typical time it takes for a commercial model to be updated, mainly due to operational costs.

B. Adaptive Stealing Attack

To countermeasure adversarial masks applied to the outputs by the victim, attackers can launch an adaptive attack based on input transformation. One notable example of such counter-attack applies the classical JPEG compression algorithm to the received image, as shown to be effective in counter-balancing the effects of adversarial attacks [28].[2] Both methods are deployed in parallel: FGSM by the victim, and JPEG as an

[2]We note that JPEG is not safe from all types of perturbations [58].

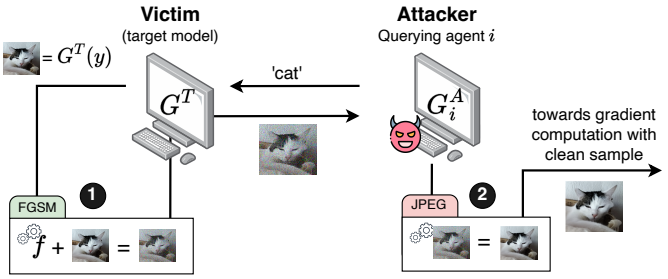

Fig. 6: FGSM defense and adaptive stealing attack: Instead of directly sending its generation $G^T(y)$ to the querying agent, the victim harnesses some model f (which is typically a publicly available classifier; in our case, LeNet-5) to craft an adversarial example ❶. As a countermeasure the attacker preventively runs a compression scheme ❷ on $G^T(y)$.

adaptive attack (see Fig. 6). To apply this adaptive attack, a thief runs through JPEG the adversarial samples received by the victim has sent him. Due to time constraints, we only show results for a subset of the parameters, *e.g.*, for $N_{q/a} = 100$, 250 and 500. Table V and Fig. 7 show these results. We observe how, in terms of FID score, results improve (overall lower FID scores), while however going higher than the application of the FGSM mask (9.14 FID lost on average for MNIST and 3.29 for FashionMNIST). The reasons are twofold: *(i)* since JPEG uses lossy compression, it lowers the quality of the sample against which the agent computes update gradients; *(ii)* FGSM introduces noise on the picture that cannot be un-made unless the mask is known, further degrading the sample quality.

A 3 : *Using a single step of FGSM, the victim was able to severely alter the quality of the stolen model's outputs. However, it did not stop the learning process since the FID of the thief improves when increasing $N_{q/a}$. Additionally, the federated thieves can recover some performance by running the adversarial samples through JPEG.*

C. Defense through sample watermarking

Studies have proposed various strategies to combat deepfakes, including the use of adversarial watermarks [34], steganography GANs [49], adaptive blind image watermarking [31], and visible watermarking using GANs [48]. These techniques demonstrate applications in scenarios where only generated images are accessible, such as verifying the ownership of GANs. Diverging from the previous sections where we used a direct defense aiming at undermining the training process of the colluding thieves, we consider a passive approach consisting in embedding an artificial fingerprint inside the victim's outputs to watermark the stolen model. For each dataset, we trained an image steganography encoder E and decoder D, using E to embed fingerprints into the training data, training a generative model with its original protocol, and finally decoding the fingerprints from the generated deepfakes. Because the model being trained in our scenario is that of the thieves, the training data being encoded are actually the query-replies of the service API, *i.e.*, the victim's outputs (see Fig. 8).

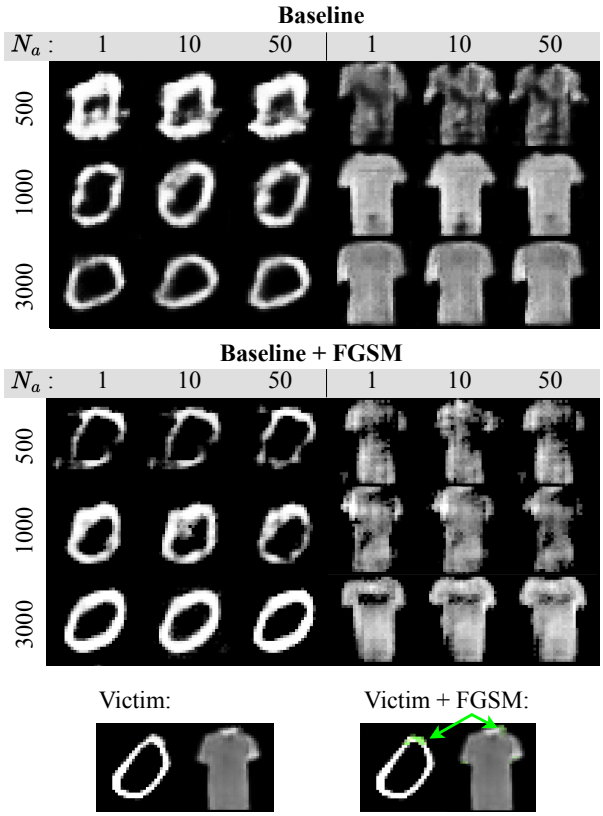

Fig. 7: Thief generated images of two classes in baseline and when the victim applies a FGSM mask on its outputs. Victim generations are shown on the lower end. For the FGSM case, the absolute magnitude of each pixel's adversarial modification is represented in green.

In practice, the decoder for the artificial fingerprints along with the fingerprinting encoder and the unique fingerprints assigned to different models, is privately maintained by the model inventor or the entity responsible for maintaining the generative model. This setup allows the model inventor to have control over the decoding process and enables them to verify the origin of generated content in case of potential misuse or attribution needs. In this case, the victim's defense is the liability of the thief generator provided by model attribution which solely consists in the ability of the decoder to recover the finger print from thief-generated-samples. Fingerprints are represented as binary vectors of length $n = 100$ as per [67]. We use bitwise accuracy between the embedded fingerprint and the one recovered by D to evaluate the detection accuracy.

For various pairs of $(N_{q/a}, N_a)$ we run CLUES 5 times with different fingerprints and queries (135 runs in total) for 900 rounds and retained the best average bitwise accuracy over 2048 samples generated by G_c^A every 100 rounds. Further details about our implementation and experiment parameters can be found in the supplementary material refered to in §IV-A. We found that the detector could never recover the embedded fingerprint with a bitwise accuracy exceeding 60% (50% is the accuracy of random guessing). A potential reason for this poor performance lies in the fact that the thief

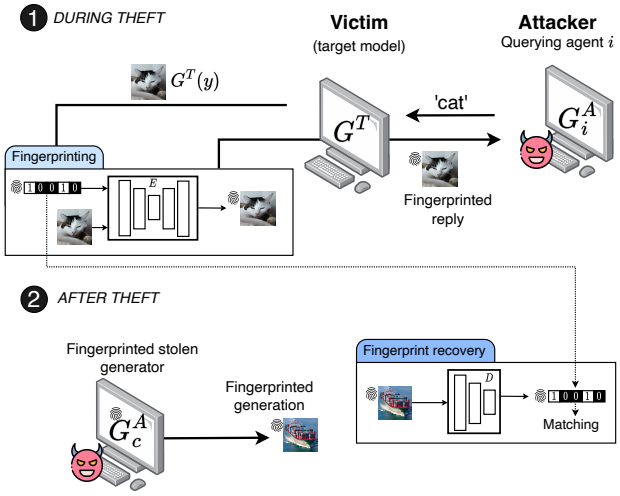

Fig. 8: Watermarking defense: (step ❶) During the stealing process, the victim-generation $G^T(y)$ is fingerprinted with an encoder E before responding to the querying user. Thieves train with fingerprinted samples (not shown) and become watermarked. Once the central attacking server reaches a satisfactory stolen model G_c^A (step ❷), it deploys it to generate samples with an embedded fingerprint unbeknownst to the thieves. The victim then feeds the samples to a decoder D, which recovers a fingerprint. This fingerprint allows for a claim of ownership over G_c^A in the event of a match.

generator is not trained in its original protocol as described in §II whereas it is seen as a prerequisite by [67].

A 4: *Using a watermarking scheme as a passive defense, the victim was not able to recover its fingerprints in the samples generated by the stolen models.*

VI. Related Work

Machine learning models publicly hosted are under continuous threat [17], [21], [55]. Several studies [52] focus on attacks and defenses against model stealing. However, model extraction attacks have been studied and demonstrated mainly on classifier models (*e.g.*, decision trees, logistic regressions, SVMs, DNNs) [35], [54]. To the best of our knowledge, we are the first to showcase effective model stealing attacks against cGANs in a distributed and federated manner. In the reminder, we summarize model stealing applied to GANs and cGANs, to other types of learning models, and other types of stealing attacks. Attempts to tackle the model stealing problem under a crypto-analytics perspective [20] is out of scope.

Stealing GANs & cGANs. A few studies demonstrate how to launch model stealing attacks against GAN models [32], [59] in a centralized manner. GANs are scrutinized against other privacy vulnerabilities. We note how membership inference attacks [22], [70] are particularly severe, in particular given how GANs are applied to domains such as medical imaging [65], where malicious users can partially reconstruct the medical history of patients. Attribute inference attacks on GAN try to infer private attributes of a data record based on other public attributes that are easily accessible. However these alone do not allow for a full model reconstruction.

Stealing other types of models. Several studies exist about stealing classifiers based on DNNs [23], [66]. These attacks assume having access to the real data to query the target classifier, achieving high fidelity and accuracy. The data-free stealing attacks [37], [72] consider more realistic scenarios where attackers need to generate the synthetic images for querying the target models. To protect against such attacks, typical defenses include prediction poisoning [53] or adversarial perturbations [69], along the same lines shown in §V. In [26], authors explain how to launch a model stealing attack against decision tree models, specifically optimized for face-recognition. Simple counter-measures [42] include a privacy-aware decision tree training algorithm, as well as revealing only rounded confidence values.

Authors of [57] demonstrate how to steal graph neural networks models or their internal links [68]. To protect against such attacks, authors show the effectiveness of deceptive perturbations [57]. Additional types of stealing attacks against diverse models exist, *e.g.*, against deep reinforcement learning models [23], collaborative inference [30], BERT-like language processing [40] or even ensemble models [44].

VII. Conclusion and Future Work

While synthetic data generation services are increasingly widespread, the concerns and risks of generative models being stolen increases. We consider a realistic adversarial scenario, with multiple model thieves colluding to steal a target victim model. We propose CluеS, the first collusive stealing strategy designed for conditional generative adversarial networks (cGANs), consisting of two phases, namely querying victim cGANs and training stolen cGANs. To train the stolen cGAN, we design an optimization algorithm that minimizes the distance of synthetic images between the victim and the stolen cGAN. We contribute a distributed model attack, where thieves train the stolen cGAN in a federated manner by aggregating local model updates periodically. We evaluate CluеS on several datasets of different degrees of complexity, demonstrating the effectiveness of our proposed collusive model stealing. Our experimental results highlight that the stolen cGANs can achieve FID scores as good as the victim cGAN. Based on our findings, we examine the efficacy of defense mechanisms in deploying generative models, while highlighting the overheads of potential counter-measures.

We plan to extend this work along these directions. First, we aim to investigate an asynchronous paradigm where the aggregating server does not wait for all thieves' local updates, potentially speeding up the outcome but at the expense of straggler nodes' computing cycles. Secondly, we will consider generators for additional data types (*i.e.*, audio, text), utilizing widely adopted diffusion-based models.

Acknowledgment

This work has been partially supported by the Spoke "FutureHPC & BigData" of the ICSC - Centro Nazionale di Ricerca in "High Performance Computing, Big Data and Quantum Computing", funded by EU - NextGenerationEU.

REFERENCES

[1] AWS Machine Learning. https://aws.amazon.com/machine-learning/. Accessed: 2024-05-02.

[2] Azure Machine Learning. https://azure.microsoft.com/en-us/products/machine-learning/. Accessed: 2024-05-02.

[3] Generative AI: AIMLAPI. https://aimlapi.com. Accessed: 2024-05-02.

[4] Generative AI: Datumbox. https://www.datumbox.com. Accessed: 2024-05-02.

[5] Generative AI: Dream. https://dream.ai/create. Accessed: 2024-05-02.

[6] Generative AI: Eden AI. https://www.edenai.co/technologies/generative-ai. Accessed: 2024-05-02.

[7] Generative AI: ImageFX. https://aitestkitchen.withgoogle.com/tools/image-fx. Accessed: 2024-05-02.

[8] Generative AI: Microsoft Designer. https://www.bing.com/images/create. Accessed: 2024-05-02.

[9] Generative AI: Replicate. https://replicate.com. Accessed: 2024-05-02.

[10] Generative AI: This X Does Not Exist. https://thisxdoesnotexist.com. Accessed: 2024-05-02.

[11] Google Cloud Generative AI. https://cloud.google.com/use-cases/generative-ai. Accessed: 2024-05-02.

[12] Information about the api. https://huggingface.co/docs/api-inference/faq. Accessed: 2024-05-02.

[13] Torchvision. https://pytorch.org/vision/stable/. Accessed: 2024-05-02.

[14] Rate limits. https://platform.openai.com/docs/guides/rate-limits?context=tier-free, Accessed: 2024-05-02.

[15] Nuha Aldausari, Arcot Sowmya, Nadine Marcus, and Gelareh Mohammadi. Video generative adversarial networks: A review. *ACM Comput. Surv.*, 55(2):30:1–30:25, 2023.

[16] Mohammed Aledhari, Rehma Razzak, Reza M. Parizi, and Fahad Saeed. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access*, 8:140699–140725, 2020.

[17] Giovanni Apruzzese, Hyrum S. Anderson, Savino Dambra, David Freeman, Fabio Pierazzi, and Kevin A. Roundy. "real attackers don't compute gradients": Bridging the gap between adversarial ML research and practice. *CoRR*, abs/2212.14315, 2022.

[18] Jean-Pierre Briot. From artificial neural networks to deep learning for music generation: history, concepts and trends. *Neural Comput. Appl.*, 33(1):39–65, 2021.

[19] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*. OpenReview.net, 2019.

[20] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In *CRYPTO (3)*, volume 12172 of *Lecture Notes in Computer Science*, pages 189–218. Springer, 2020.

[21] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. Exploring connections between active learning and model extraction. In *USENIX Security Symposium*, pages 1309–1326. USENIX Association, 2020.

[22] Dingfan Chen, Ning Yu, Yang Zhang, and Mario Fritz. Gan-leaks: A taxonomy of membership inference attacks against generative models. In *CCS*, pages 343–362. ACM, 2020.

[23] Kangjie Chen, Shangwei Guo, Tianwei Zhang, Xiaofei Xie, and Yang Liu. Stealing deep reinforcement learning models for fun and profit. In *AsiaCCS*, pages 307–319. ACM, 2021.

[24] Li Deng. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process. Mag.*, 29(6):141–142, 2012.

[25] Donatella Firmani, Francesco Leotta, and Massimo Mecella. On computing throttling rate limits in web apis through statistical inference. In *ICWS*, pages 418–425. IEEE, 2019.

[26] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *CCS*, pages 1322–1333. ACM, 2015.

[27] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR (Poster)*, 2015.

[28] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering adversarial images using input transformations. In *ICLR (Poster)*. OpenReview.net, 2018.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034. IEEE Computer Society, 2015.

[30] Zecheng He, Tianwei Zhang, and Ruby B. Lee. Model inversion attacks against collaborative inference. In *ACSAC*, pages 148–162. ACM, 2019.

[31] Ling-Yuan Hsu. Ai-assisted deepfake detection using adaptive blind image watermarking. *Journal of Visual Communication and Image Representation*, 100:104094, 2024.

[32] Hailong Hu and Jun Pang. Stealing machine learning models: Attacks and countermeasures for generative adversarial networks. In *ACSAC*, pages 1–16. ACM, 2021.

[33] Xia Hu, Lingyang Chu, Jian Pei, Weiqing Liu, and Jiang Bian. Model complexity of deep learning: a survey. *Knowl. Inf. Syst.*, 63(10):2585–2619, 2021.

[34] Hao Huang, Yongtao Wang, Zhaoyu Chen, Yuze Zhang, Yuheng Li, Zhi Tang, Wei Chu, Jingdong Chen, Weisi Lin, and Kai-Kuang Ma. Cmua-watermark: A cross-model universal adversarial watermark for combating deepfakes. In *AAAI*, pages 989–997. AAAI Press, 2022.

[35] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *USENIX Security Symposium*, pages 1345–1362. USENIX Association, 2020.

[36] Jinsung Jeon, Jeonghak Kim, Haryong Song, Seunghyeon Cho, and Noseong Park. GT-GAN: general purpose time series synthesis with generative adversarial networks. In *NeurIPS*, 2022.

[37] Sanjay Kariyappa, Atul Prakash, and Moinuddin K. Qureshi. MAZE: data-free model stealing attack using zeroth-order gradient estimation. In *CVPR*, pages 13814–13823. Computer Vision Foundation / IEEE, 2021.

[38] Byungjoo Kim, Suyoung Lee, Seanie Lee, Sooel Son, and Sung Ju Hwang. Margin-based neural network watermarking. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 16696–16711. PMLR, 2023.

[39] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016.

[40] Kalpesh Krishna, Gaurav Singh Tomar, Ankur P. Parikh, Nicolas Papernot, and Mohit Iyyer. Thieves on sesame street! model extraction of bert-based apis. In *ICLR*. OpenReview.net, 2020.

[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[42] Taesung Lee, Benjamin Edwards, Ian M. Molloy, and Dong Su. Defending against neural network model stealing attacks using deceptive perturbations. In *IEEE Symposium on Security and Privacy Workshops*, pages 43–49. IEEE, 2019.

[43] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, pages 3730–3738. IEEE Computer Society, 2015.

[44] Zhuo Ma, Xinjing Liu, Yang Liu, Ximeng Liu, Zhan Qin, and Kui Ren. Divtheft: An ensemble model stealing attack by divide-and-conquer. *IEEE Trans. Dependable Secur. Comput.*, 20(6):4810–4822, 2023.

[45] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[46] Fabian Mentzer, George Toderici, David Minnen, Sergi Caelles, Sung Jin Hwang, Mario Lucic, and Eirikur Agustsson. VCT: A video compression transformer. In *NeurIPS*, 2022.

[47] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[48] Aakash Varma Nadimpalli and Ajita Rattani. ProActive DeepFake Detection using GAN-based Visible Watermarking. *ACM Trans. Multimedia Comput. Commun. Appl.*, sep 2023.

[49] Iram Noreen, Muhammad Shahid Muneer, and Saira Andleeb Gillani. Deepfake attack prevention using steganography gans. *PeerJ Comput. Sci.*, 8:e1125, 2022.

[50] Artem Obukhov and Mikhail Krasnyanskiy. Quality assessment method for gan based on modified metrics inception score and fréchet inception distance. In Radek Silhavy, Petr Silhavy, and Zdenka Prokopova, editors, *Software Engineering Perspectives in Intelligent Systems*, pages 102–114, Cham, 2020. Springer International Publishing.

[51] Seong Joon Oh, Bernt Schiele, and Mario Fritz. Towards reverse-engineering black-box neural networks. In *Explainable AI*, volume 11700 of *Lecture Notes in Computer Science*, pages 121–144. Springer, 2019.

[52] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Comput. Surv.*, 55(14s):324:1–324:41, 2023.

[53] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Prediction poisoning: Towards defenses against DNN model stealing attacks. In *ICLR*. OpenReview.net, 2020.

[54] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *AsiaCCS*, pages 506–519. ACM, 2017.

[55] Maria Rigaki and Sebastian García. A survey of privacy attacks in machine learning. *ACM Comput. Surv.*, 56(4):101:1–101:34, 2024.

[56] Sagar Sharma and Keke Chen. Confidential machine learning on untrusted platforms: a survey. *Cybersecur.*, 4(1):30, 2021.

[57] Yun Shen, Xinlei He, Yufei Han, and Yang Zhang. Model stealing attacks against inductive graph neural networks. In *SP*, pages 1175–1192. IEEE, 2022.

[58] Richard Shin and Dawn Song et al. JPEG-resistant Adversarial Images. In *Proceedings of the Workshop on Machine Learning and Computer Security at NIPS 2017*, January 2018. [Online; accessed 4. Dec. 2023].

[59] Hui Sun, Tianqing Zhu, Zhiqiu Zhang, Dawei Jin, Ping Xiong, and Wanlei Zhou. Adversarial attacks against deep generative models on data: A survey. *IEEE Trans. Knowl. Data Eng.*, 35(4):3367–3388, 2023.

[60] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826. IEEE Computer Society, 2016.

[61] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant gans: Deep generation of financial time series. *CoRR*, abs/1907.06673, 2019.

[62] Xian Wu, Kun Xu, and Peter et al. Hall. A survey of image synthesis and editing with generative adversarial networks. *Tsinghua Science and Technology*, 22(6):660–674, 2017.

[63] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[64] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional GAN. In *NeurIPS*, pages 7333–7343, 2019.

[65] Xin Yi, Ekta Walia, and Paul S. Babyn. Generative adversarial network in medical imaging: A review. *Medical Image Anal.*, 58, 2019.

[66] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. Cloudleak: Large-scale deep learning models stealing through adversarial examples. In *NDSS*. The Internet Society, 2020.

[67] Ning Yu, Vladislav Skripniuk, Sahar Abdelnabi, and Mario Fritz. Artificial fingerprinting for generative models: Rooting deepfake attribution in training data. In *ICCV*, pages 14428–14437. IEEE, 2021.

[68] He Zhang, Bang Wu, Shuo Wang, Xiangwen Yang, Minhui Xue, Shirui Pan, and Xingliang Yuan. Demystifying uneven vulnerability of link stealing attacks against graph neural networks. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 41737–41752. PMLR, 2023.

[69] Jiliang Zhang, Shuang Peng, Yansong Gao, Zhi Zhang, and Qinghui Hong. APMSA: adversarial perturbation against model stealing attacks. *IEEE Trans. Inf. Forensics Secur.*, 18:1667–1679, 2023.

[70] Minxing Zhang, Ning Yu, Rui Wen, Michael Backes, and Yang Zhang. Generated distributions are all you need for membership inference attacks against generative models. In *WACV*, pages 4827–4837. IEEE, 2024.

[71] Zilong Zhao, Aditya Kunar, Robert Birke, and Lydia Y. Chen. CTAB-GAN: effective table data synthesizing. In *ACML*, volume 157 of *Proceedings of Machine Learning Research*, pages 97–112. PMLR, 2021.

[72] Mingyi Zhou, Jing Wu, Yipeng Liu, Shuaicheng Liu, and Ce Zhu. Dast: Data-free substitute training for adversarial attacks. In *CVPR*, pages 231–240. Computer Vision Foundation / IEEE, 2020.