Generating High-fidelity, Synthetic Time Series Datasets with DoppelGANger

Zinan Lin
Carnegie Mellon University
zinanl@andrew.cmu.edu

Alankar Jain
Carnegie Mellon University
alankarj@andrew.cmu.edu

Chen Wang
IBM
Chen.Wang1@ibm.com

Giulia Fanti
Carnegie Mellon University
gfanti@andrew.cmu.edu

Vyas Sekar
Carnegie Mellon University
vsekar@andrew.cmu.edu

Abstract

Limited data access is a substantial barrier to data-driven networking research and development. Although many organizations are motivated to share data, privacy concerns often prevent the sharing of proprietary data, including between teams in the same organization and with outside stakeholders (e.g., researchers, vendors). Many researchers have therefore proposed synthetic data models, most of which have not gained traction because of their narrow scope. In this work, we present DoppelGANger, a synthetic data generation framework based on generative adversarial networks (GANs). DoppelGANger is designed to work on time series datasets with both continuous features (e.g. traffic measurements) and discrete ones (e.g., protocol name). Modeling time series and mixed-type data is known to be difficult; DoppelGANger circumvents these problems through a new conditional architecture that isolates the generation of metadata from time series, but uses metadata to strongly influence time series generation. We demonstrate the efficacy of DoppelGANger on three real-world datasets. We show that DoppelGANger achieves up to 43% better fidelity than baseline models, and captures structural properties of data that baseline methods are unable to learn. Additionally, it gives data holders an easy mechanism for protecting attributes of their data without substantial loss of data utility.

1 Introduction

Data-driven research is a centerpiece of networking and systems research and development, as well as many other fields [9, 14, 17, 35, 46, 46, 59, 60, 67, 85]. Realistic datasets allow engineers to build a better understanding of existing systems, motivate design choices from actual needs, and prove that proposed new systems can indeed work in practice.

Unfortunately, the potential benefits of data-driven research and development have been somewhat restricted: generally, only select players who possess data can reliably perform research or develop products. Many of these players are reluctant to share datasets for fear of revealing business secrets, running afoul of data regulations, or violating customers' privacy; this is certainly true of data sharing for academic research, but it is also often true of sharing data across teams within the same organization, as well as business partnerships (e.g., vendors). Notable exceptions aside (e.g., [1, 62]), the issue of data access has been and continues to be a substantial concern in the networking and systems communities.

An appealing alternative is to create synthetic datasets that can be safely released to enable research and cross-stakeholder collaboration. Such datasets should ideally have three (sometimes conflicting) properties:

Fidelity: The synthetic data should be drawn from the same (or a similar) distribution to an underlying real dataset.

Flexibility: The models should allow researchers to generate different classes of data. For example, we may wish to augment the amount of data representing anomalous or sparse events such as hardware failures or flash crowds.

Privacy: The data release technique should be compatible with anonymization and/or data privatization techniques that do not destroy the utility of the data [8, 53, 68, 72, 86].

In this paper, we consider an important and broad class of networking/systems datasets—time series measurements of multi-dimensional features, associated with multi-dimensional attributes. Many networking and systems datasets fall in this category, including traffic traces [1, 76, 90], measurements of physical network properties [9, 10], and datacenter/compute cluster usage measurements [13, 41, 73]. For example, measurements from a compute cluster might include a separate time series for each task measuring per-epoch CPU usage, memory usage, as well as categorical attributes like the exit code of the task (e.g., KILL, FAIL, FINISH).

While there have been decades of work on building synthetic data models for time series data, including for networking and systems applications (§2.2), existing solutions typically provide only a subset of the desired properties. For

Approach	**Flexibility**	**Privacy**	**Fidelity**
raw data	no	no	best
anonymized raw data	no	?	good
Markov model	yes	yes	bad
autoregressive model	yes	yes	bad
RNN	yes	yes	bad
GANs	**yes**	**yes**	**good**

Table 1: The potential of GANs to satisfy key desirable properties of synthetic datasets.

example, anonymized data can provide high fidelity but is susceptible to common privacy concerns, including membership inference attacks (§5.3.1), leakage of sensitive attributes (§5.3.2), and even deanonymization [8, 68, 72, 86]. Even combinations of these baselines do not solve the problem; we show in §5.1 that all of the generative model baselines have inadequate fidelity on time series data, whereas anonymized raw data has inadequate privacy, so combinations thereof will exhibit at least one of these weaknesses.

In this paper, we explore if and how we can leverage recent advances in the space of Generative Adversarial Networks (GANs) [33] to facilitate data-driven network research. GANs have spurred much excitement in the machine learning community due to their ability to generate photorealistic images [49], previously considered a challenging problem. This suggests the potential promise of using it to synthesize high-fidelity networking and systems datasets that stakeholders can release to the research community and/or commercial data users, either inside or outside the stakeholders' enterprise. We compare the potential of GAN-based solutions with other baselines in Table 1.

However, we find that naive GAN implementations are unable to model the data with high fidelity because of the following challenges: (1) Complex correlations between time series and their associated attributes. For instance, in a cluster dataset [73], as the memory usage of a task increases over time, its likelihood of failure increases. Existing GAN architectures do not learn such correlations well. (2) Long-term correlations within time series, such as diurnal patterns. These correlations are qualitatively very different from those found in images, which have a fixed dimension and do not need to be generated pixel-by-pixel. Indeed, existing GANs struggle to generate long time series.

Our main algorithmic contribution is a GAN-based framework called DoppelGANger that addresses both challenges. Key features include:

(1) Decoupled attribution generation: To learn better correlations between time series and their attributes (e.g., metadata like ISP name or location), DoppelGANger decouples the generation of attributes from time series and feeds attributes to the time series generator *at each time step*. This contrasts with conventional approaches, where attributes and features are generated jointly. This conditional generation architecture also offers us the flexibility to change the attribute distribution without sacrificing fidelity and enables us to hide the real attribute distribution when it is a privacy concern.

(2) Batched generation: To strengthen temporal correlations in time series, DoppelGANger outputs *batched* samples rather than singletons. This idea has been used widely in Markov modeling [32], but its effects on GANs is still an active research topic [56, 75] that has not been studied in the context of time series generation (to the best of our knowledge).

(3) Decoupled normalization: We observe that traditional GANs trained on datasets with a highly variable dynamic range across samples tend to exhibit severe mode collapse, where the generator always outputs very similar samples. We believe this phenomenon has not yet been documented in the GAN literature, which typically experiments with a narrow class of signals (e.g., images); in contrast, networking time series exhibit much more variability across each sample's max/min limits. To address this, our architecture separately generates normalized time series and realistic max and min limits conditioned on the sample attributes.

We evaluate DoppelGANger on three real-world datasets, including web traffic time series [34], geographically-distributed broadband measurements [20], and compute cluster usage measurements [73]. To demonstrate fidelity on these datasets, we first show that DoppelGANger is able to learn structural microbenchmarks of each dataset better than baseline approaches. We use this exploration to systematically evaluate how each component in DoppelGANger affects performance, and provide recommended hyper-parameter choices. We then test DoppelGANger-generated data on downstream tasks, such as training prediction algorithms. We find that DoppelGANger consistently outperforms baseline algorithms; predictors trained on DoppelGANger-generated data have test accuracies on real data that are up to 43% higher than when trained on baseline-generated data. We also highlight how DoppelGANger allows end users the flexibility to re-train models to emphasize certain classes of data.

Our results on privacy are mixed and suggest more active research is needed. On the positive side, we show that DoppelGANger is able to seamlessly obfuscate the distribution of sensitive data attributes without sacrificing utility; this task was specifically highlighted as a privacy concern by a company we talked with. Similarly, we find that an important class of *membership inference* attacks on privacy can be mitigated by training DoppelGANger on larger datasets (§5.3.1). This counters conventional data release practices, which advocate releasing smaller datasets to avoid leaking user data [74]. That said, DoppelGANger does not fully solve the privacy problem. To our surprise, we find that recently-proposed techniques for training GANs with *differential privacy* guarantees [2, 12, 24] have a poor fidelity-privacy trade-off on our datasets, almost completely destroying temporal correlations for moderate privacy guarantees (§5.3.1). This suggests that existing differential privacy machine learning techniques may be inadequate for networking applications and require further research.

This work is only a first step towards using GANs to generate realistic and privacy-preserving synthetic data. For instance, as mentioned above, the anonymity and privacy properties of GAN-generated data leave much room for future exploration. Nonetheless, we view it as an important first step to demonstrate that GANs can achieve acceptable fidelity and basic privacy on a broad class of datasets, and can thus serve as a basis for broadening the benefits and opportunities of data-driven network design and modeling.

2 Motivation and Related Work

In this section, we highlight some motivating scenarios and why existing solutions fail to achieve our goals.

2.1 Use cases

While there are many potential use cases that can benefit from a system like DoppelGANger, we highlight two key types of interactions and three representative tasks in these cases:
Stakeholder interactions: There are two natural scenarios:

- *Collaboration across enterprises:* Enterprises often impose restrictions on data access between their own divisions and/or with external vendors due to privacy concerns. DoppelGANger can be used to share representative data models between collaborators without violating privacy restrictions on user data.
- *Reproducible, open research:* Many research ideas rely on access to proprietary datasets on which to test and develop new ideas. However, the provider's policies and/or business considerations may preclude these datasets from being available and thus render the resulting research irreproducible. If the data providers could release a DoppelGANger model of the shared dataset, researchers could independently reproduce results without requiring access to user data.

Data-driven tasks: In such interactions, we can consider three representative *tasks*:

1. *Algorithm design:* The design of many resource allocation algorithms such as cluster scheduling, resource allocation, and transport protocol design (e.g., [17, 35, 46, 59, 60, 67]) often needs workload data to tune control parameters. As such, a key property for generated data is that if algorithm A performs better than algorithm B on the real data, then the same should hold on the generated data.
2. *Structural characterization:* Many system designers also need to understand structural temporal and/or geographic trends in systems; e.g., to understand the shortcomings and/or resource mismatches in existing systems and to suggest remedial solutions [9, 14, 46, 85]. In this case, generated data should preserve trends and distributions well enough to reveal such structural insights.
3. *Predictive modeling:* A third use case for time series data is to learn predictive models, especially for rare or anomalous events such as network anomalies [31, 47, 55]. For these models to be useful, they should have enough fidelity that a predictor trained on generated data should make meaningful predictions on real data.

These use cases highlight the need for models that exhibit fidelity, privacy, and flexibility. Without fidelity, data recipients cannot draw meaningful conclusions (or may draw incorrect ones). Without privacy, the data provider may be liable for violations of privacy policies. Without flexibility, the data recipients may be limited in the kinds of experiments or analytics they can run.

2.2 Related work and limitations

Our focus in this work is on *multi-dimensional time series* datasets, which are common in networking and systems applications. Examples include: *1. Web traffic traces* of temporal web page views with attributes of web page names that can be used to predict future daily views, analyze page correlations [84], or generate page recommendations [28, 69]; 2. *Network measurements* of packet loss rate, bandwidth, delay from Internet-connected devices with attributes such as location or device type that are useful for network management [47]; or *3. Cluster usage measurements* of metrics such as CPU/memory usage associated with attributes (e.g., server and job type) that can inform resource provisioning [15] and job scheduling [61]. Each of the listed examples consists of time series data with (potentially) high-dimensional data points and associated attributes (metadata) that can be either numeric or categorical.

Generative models for such time series have a rich literature (e.g., [42, 58, 63–66, 80, 89]). However, primary shortcomings of prior work are: low fidelity, low flexibility, and/or requiring detailed domain knowledge. For example, prior works on network trace generation assume parametric models for entities like networks, users, traffic patterns, and file sizes, and use data to infer those parameters [42, 80, 89]. Since we do not limit ourselves to network traces, we want to avoid the time-intensive task of deriving parametric physical models for many possible data sources (e.g., compute cluster, WAN, web ecosystem).

At a high level, most prior efforts for modeling time series data in the networking and systems community are based on one of the following statistical models:
Dynamic stationary processes represent each point in the time series $(R_i, i \geq 0)$ as $R_i = X_i + W_i$, where X_i is a deterministic process, and W_i is a noise function. This is a widely used approach for modeling traffic time series in the networking community [4, 52, 63–66, 70, 81]. Some of these efforts rely

critically on *a priori* knowledge of key patterns (e.g., diurnal trends) to constrain the deterministic process, while also using naive noise models (e.g., Gaussian). This is infeasible for modeling datasets with complex, unknown correlations. Others, such as the *Transform–Expand–Sample (TES)* methodology [63–66], instead use an empirical estimate of each time series' marginal distribution *and* autocorrelation (assuming stationarity). Unfortunately, empirical histograms are known to be poor estimates of high-dimensional distributions [77].

Markov models (MMs) are a popular approach for modeling categorical time series, by representing system dynamics as a conditional probability distribution satisfying $P(R_i|R_{i-1},\dots,R_1) = P(R_i|R_{i-1})$. Variants such hidden markov models [27]) from the text generation literature [21,45] have also been used for modeling the distributions of time series [32,39,54]. More recently, neural network-based approaches have been shown to outperform Markov models in many settings [44,51]. The key weakness of MMs is their inability to encode long-term correlations in data.

Auto-regressive (AR) models improve on dynamic stationary processes for time series modeling [25]. In AR models, each point in the time series (R_i) is represented as a function of the previous p points: $R_i = f(R_{i-1}, R_{i-2},\dots,R_{i-p}) + W_i$, where W_i is white noise. *Nonlinear AR models* (e.g., parameterized by neural networks) have gained traction and are the baseline used in this work [3,88,95,96]. The main problem with AR models is fidelity: like Markov models, they only use a limited history to predict the next sample in a time series, leading to over-simplified temporal correlations.

Recurrent neural networks (RNNs) have been more recently used for time series modeling in deep learning [43]. Like AR and Markov models, they use the previous sample to determine the next sample, but RNNs also store an internal state variable that captures the entire history of the time series. RNNs have had great success in learning *discriminative* models of time series data, which predict a label conditioned on a sample [44,51]. However, generative modeling is harder than discriminative modeling. Indeed, we find RNNs are unable to learn certain simple time series distributions.

GAN-based methods GANs have emerged as a popular technique for generating or augmenting datasets, especially medical images or patient records [19,29,36,38,78]. As discussed in §3.3, the architectures used for those tasks give poor fidelity in networking data, which has both complex temporal correlations and mixed discrete-continuous data types. Although GAN-based time series generation exists (e.g., text [22,98], medical time series [12,24]) we find that such techniques fail on networking data, exhibiting poor fidelity on longer sequences and severe mode collapse. This is partially because networking data tends to be more heavy-tailed and variable in length than medical time series, which seems to affect GANs in ways that have not been carefully explored before.

In summary, prior approaches fail to capture long- and/or short-term temporal correlations, or do so only with extensive

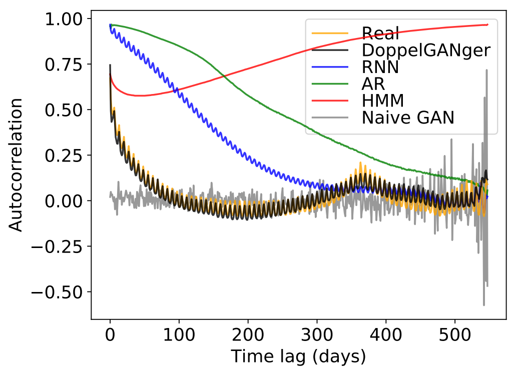

Figure 1: Autocorrelation of daily page views for Wikipedia Web Traffic dataset. DoppelGANger captures both weekly and annual correlation pattern.

prior knowledge. To illustrate this, we run the above baselines on the Wikipedia Web Traffic dataset, which contains daily page views of web pages (details in §5). Figure 1 shows the autocorrelation of time series samples (averaged over all samples) for real and synthetic datasets. Two patterns emerge in the real data: a short-term weekly correlation pattern indicated by the periodic spikes in autocorrelation and a long-term annual correlation indicated by the local peak at roughly the 1-year mark (365 days). Notably, all of our baselines fail to capture both patterns simultaneously. HMMs (purple line) and AR models (red line) do not store enough states to learn even the weekly correlations, and the state space required to learn long-term correlations would be prohibitively large. RNNs learn the short-term correlation correctly, but do not capture the long-term correlation. Even naive GANs (§3.3) fail to learn a meaningful autocorrelation.

3 Problem Statement and Scope

We abstract our datasets as follows: A *dataset* $\mathcal{D} = \{O^1, O^2, ..., O^n\}$ is defined as a set of *objects* O^i. Each object represents an atomic, high-dimensional data element (i.e., the combination of a single time series with its associated metadata). More precisely, each object $O^i = (A^i, R^i)$ contains *m attributes* $A^i = [A^i_1, A^i_2, ..., A^i_m]$. For example, attribute A^i_j could represent user i's physical location, and A^i_k for $k \neq j$ the user's ISP. Note that we can support datasets in which multiple objects have the same set of attributes. The second component of each object is a time series of *records* $R^i = [R^i_1, R^i_2, ..., R^i_{T^i}]$. For example, in a network trace dataset, the time series might contain the packets sent out from a specific client; each packet consists of a single record. Different objects may contain different numbers of records (i.e., time series of different lengths). The number of records for object O^i is given by T^i. Each record $R^i_j = (t^i_j, f^i_j)$ contains a *timestamp* t^i_j, and K *features* $f^i_j = [f^i_{j,1}, f^i_{j,2}, ..., f^i_{j,K}]$ (e.g. the features of the packet). Note that the timestamps are sorted, i.e.

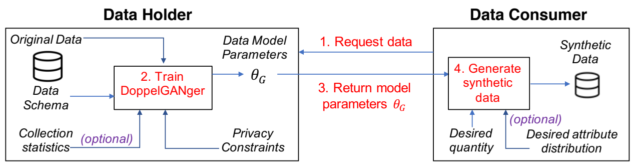

Figure 2: Workflow for DoppelGANger usage.

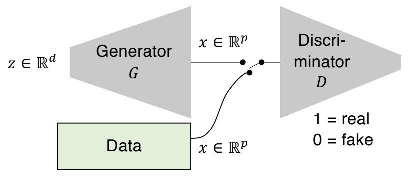

Figure 3: Original GAN architecture from [33].

$t_j^i < t_{j+1}^i \; \forall 1 \leq j < T^i$. In our work, we treat the timestamps as equally spaced and hence do not generate them explicitly. However we can easily extend this to unequally spaced timestamps by treating time as a continuous feature and generating inter-arrival times along with other features.

This abstraction fits many classes of data that appear in networking applications. For example, Table 6 maps a web traffic measurement dataset to our terminology. It does not apply to other classes of networking and systems datasets, including lists of items (e.g., blacklisted domains), and one-shot measurements (e.g., network topology). Our problem is to take any such dataset as input and learn a model that can generate a new dataset $\mathcal{D}'$ as output. $\mathcal{D}'$ should exhibit fidelity, flexibility, and privacy, and the methodology should be general enough to handle datasets in our abstraction.

3.1 Interface

Our intended interface for DoppelGANger users is illustrated in Figure 2. DoppelGANger requires a small amount of tuning enabled by a few auxiliary inputs:

Data schema: DoppelGANger needs several properties of the data schema, including attribute/feature dimensionality, and whether they are categorical or numeric.

Time series collection frequency: Although this input is optional, we show in §5 that DoppelGANger benefits from knowing the timescale at which data was collected (e.g., minutes, seconds, days) and the total time series duration.

Privacy constraints: Data holders should input a list of sensitive attributes, whose distribution can be masked.

Once the model is trained and released, the client can generate a synthetic dataset with the following inputs:

Desired data quantity: The client can generate as much synthetic data as desired.

Desired attribute distribution: The client can optionally specify which attributes should be present in the generated dataset, and with what distribution. This facilitates exploration of anomalous events from the original data, e.g., flash crowds.

3.2 GANs: Background and promise

GANs are a data-driven generative modeling technique based on adversarial training [33]. GANs take as input a set of training data samples and output a model that can produce new samples from the same distribution as the original data, *without* simply sampling the initial dataset. GANs are seen as a breakthrough in generative modeling for their ability to generate photorealistic images [49], previously considered a difficult task due to the complex correlations in images.

More precisely, suppose we have a dataset consisting of n samples (or *objects*, in the language of our abstraction) $O_1, \ldots, O_n$, where $O_i \in \mathbb{R}^p$, and each sample is drawn i.i.d. from some distribution $O_i \sim P_O$. The goal of GANs is to use these samples to learn a model that can draw samples from distribution P_O [33]. The core idea of GANs is to train two components: a generator G and a discriminator D (Figure 3); in practice, both are instantiated with neural networks. In the original GAN design, the generator maps a noise vector $z \in \mathbb{R}^d$ to a sample $O \in \mathbb{R}^p$, where $p \gg d$. z is drawn from some pre-specified distribution P_z, usually a Gaussian. Simultaneously, we train the discriminator $D : \mathbb{R}^p \to [0,1]$, which takes samples as input (either real of fake), and classifies each sample as real (1) or fake (0).

Errors in this classification task are used to train the parameters of both the generator and discriminator through backpropagation. The loss function for GANs can be written

$$\min_G \max_D \mathbb{E}_{x \sim p_x}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]. \quad (1)$$

The generator and discriminator are trained alternately, or *adversarially*. Unlike prior generative modeling approaches, which typically involve likelihood maximization of parametric models (e.g., §2.2), GANs train models with fewer assumptions about data structure, and may be better-suited to the generative modeling of time series data than baselines.

3.3 Challenges

Despite GANs' success at modeling images, naively applying GANs to networking and systems timeseries datasets gives poor results. To show this, we implemented the first GAN architecture one might think of. The key aspects of a GAN's design are the generator architecture, the discriminator architecture, and the loss function. For this test, we used a fully-connected multilayer perceptron (MLP) generator which generates features and attributes jointly, an MLP discriminator, and Wasserstein loss.[1] Details of this experiment can be found in Appendix B. We train our naive GAN on

[1]Wasserstein loss has been widely shown to give better stability and mode coverage than the original cross-entropy loss, and it is now common practice to start with Wasserstein loss in GAN designs [24, 37, 49]. These design choices are consistent with prior work on GAN-based synthetic data generation [19, 29, 36, 38].

the Wikipedia Web Traffic dataset. The gray curve in Figure 1 shows the autocorrelation of generated samples from this naive GAN, which capture neither weekly nor annual correlations. Intuitively, this happens because naive architectures cannot generate long time series; during training, the MLP must jointly learn all cross-correlations and cannot exploit the fact that patterns often recur in time series. Moreover, this architecture does not learn the correlations between features and attributes, and it does not allow flexible generation of time series conditioned on attributes; in fact, training does not even reliably converge. Additional evaluations are in §5. Next, we describe how DoppelGANger addresses these problems.

4 Detailed Design

Our naive GAN experiments highlight the difficulty of learning long temporal correlations and correlations between features and attributes. During these experiments, we also find that mode collapse—a phenomenon where a trained GAN outputs homogenous samples despite being trained on a diverse dataset [56,83]—can happen even if we use Wasserstein loss.[2] In this section, we discuss design choices in the generator (§4.1), discriminator (§4.2), and loss function (§4.3) that address these challenges; some of these choices are unconventional, emerging precisely because of the class of problems we consider. We summarize the big picture design in §4.4.

4.1 Generator

Our generator is designed to address three key problems: capturing temporal correlations, capturing correlations between features and attributes, and alleviating mode collapse.

4.1.1 Temporal correlations

DoppelGANger learns temporal correlations by both using RNNs in the generator and generating batched data samples. The generator architecture is shown in Figure 6.

RNN generator: One reason the naive architecture fails is that MLPs are too weak to capture long-term correlations. As mentioned in §2.2, RNNs are specifically designed to model time series. RNNs generate one record R^i_j (e.g., the CPU rate at a particular second) at a time, and take T^i passes to generate the entire time series. Unlike traditional neural units, RNN neural units have an internal state that is meant to encode all past states of the signal, so that when generating R^i_j, the RNN unit can incorporate the patterns in $R^i_1,\ldots,R^i_{j-1}$. We use a widely-used RNN called long short-term memory (LSTM) [43], which is known to memorize history well.

Note that the output of an RNN has fixed dimensionality, depending on the number of RNN units. To generate data with the desired dimensionality and data types (categorical vs. continuous), we append an MLP network to the output of the RNN with the desired dimensionality. This is how we use the data schema from §3.1 Also, notice that the RNN is a single neural network, even though Figure 6 suggests that there are many units. This *unrolled* representation commonly conveys that the RNN is being used many times to generate samples.

[2]Wasserstein loss is believed to help with mode collapse [6,37].

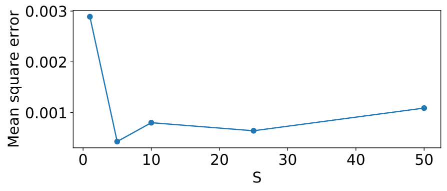

Figure 4: Batching parameter S vs the MSE of generated and real sample autocorrelations on the WWT dataset.

Generating batched points: Even with an RNN generator, GANs still struggle to capture temporal correlations when the time series length exceeds about 500. We observe a similar phenomenon in prior work using GANs to generate text [26,94]. Improving the learning capability of RNNs is an important research area [16,48] beyond the scope of our paper. Instead of entirely relying on RNNs to capture temporal correlations, we seek a tradeoff between the utilization of RNNs and MLPs. Specifically, at the generation step j, the MLP network reads the output from the RNN and generates a *batch* of samples $R^i_j, R^i_{j+1},\ldots,R^i_{j+S}$, where S is a tunable parameter. For example, if the time series length is 500 and $S = 10$, then the RNN only needs to iterate 50 times, and at each time the MLP outputs 10 consecutive records. This way, we decrease the difficulty of learning for the RNN, and exploit MLPs in a domain where they perform well. For example, Figure 4 shows the mean square error between the autocorrelation of our generated signals and real data on the WWT dataset. Even using a small (but larger than 1) batch size gives substantial improvements in signal quality (more plots in Appendix G).

Generation flag for variable length: Besides capturing temporal correlations, another challenge is that time series may have different lengths. To generate samples with variable lengths, we add a *generation flag* that indicates whether the time series has ended or not: if the time series does not end at this time step, the generation flag is $[1,0]$; if the time series ends exactly at this time step, the generation flag is $[0,1]$. We pad the rest of the time series (including all features) with 0's. The generator outputs generation flag $[p_1, p_2]$ through a softmax output layer, so that $p_1, p_2 \in [0,1]$ and $p_1 + p_2 = 1$. $[p_1, p_2]$ is used to determine whether we should continue unrolling the RNN to the next time step. One way to interpret this is that p_1 gives the probability that the time series ends at this time step. Therefore, if $p_1 < p_2$, we stop generation and pad all future features with 0's; if $p_1 > p_2$, we continue unrolling the RNN to generate features for the next time step(s). The generation flags are also fed to the discriminator (§4.2) as part of the features, so the discriminator can also learn sample length characteristics.

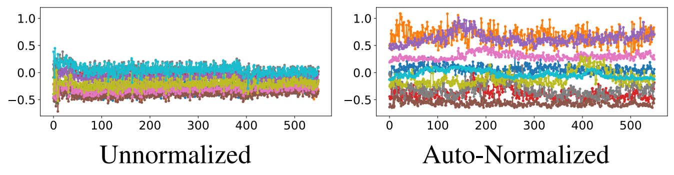

Figure 5: Ten random, synthetic WWT data samples generated by DoppelGANger with a specific attribute before and after adding auto-normalization to mitigate mode collapse.

4.1.2 Correlations between features and attributes

Recall that naive GANs fail to capture the correlation between features R^i and attributes A^i when the data objects $O^i = (A^i, R^i)$ are generated at once. To address this, we partition the data object generation into two phases: first generate the attributes, then features conditioned on the attributes.

Decoupling feature and attribute generation: Our decoupled architecture factorizes the data distribution as follows:

$$P(O^i) = P(A^i, R^i) = P(A^i) \cdot P(R^i | A^i)$$

For attribute generation, we use a dedicated MLP network that maps an input noise vector to attribute A^i. For feature generation, we use the RNN-based architecture from §4.1.1. To explicitly encourage learning of the attribute-time series correlation, we feed the generated A^i as an input to the RNN at every step. This design choice separates the hard problem into two easier sub-problems, each solved by an appropriate network architecture. Experiments show that this architecture successfully captures the correlation between features and attributes (§5.1).

Additionally, model users may want to augment the number of samples from a particular set of attributes to simulate rare events (flexibility). In other cases, users may want to hide the attribute distribution, (e.g., hardware types in a compute cluster) when it represents a business secret (privacy). This design allows users to change attribute distributions without hurting the conditional distribution or temporal correlations, by training on the original dataset and then retraining only the attribute generation MLP network to a different, desired distribution. Hence, in addition to fidelity benefits, this architecture also aids in providing flexibility and privacy.

4.1.3 Alleviating mode collapse

We find that when the range of feature values vary widely across samples in the training data (e.g., one web page has 1k−5k daily page views, while the other has only 0−100 daily page views), naive implementations exihibit severe mode collapse (Figure 5, left). This cause of mode collapse is undocumented in the GAN literature, to the best of our knowledge. One possible reason is because in natural images—the most widely-used data type in the GAN literature—pixel ranges are similar, whereas networking data tends to exhibit more dramatic fluctuations. We experimented with known state-of-the-art techniques for mitigating mode collapse [56], but found that they did not fully resolve the problem.

Auto-normalization: To prevent this mode collapse, we normalize the real data features prior to training and add $(\max\{f^i\} \pm \min\{f^i\})/2$ as two additional attributes to the i-th sample. In the generated data, we can use these two attributes to scale features back to a realistic range. However, if we generate these two *fake* attributes along with the real ones, we lose the ability to condition feature generation on a particular attribute in §4.1.2. Therefore, we further divide the generation into three steps: (1) generate attributes using the MLP generator (same as §4.1.2); (2) with the generated attributes as inputs, generate the two “fake" (max/min) attributes using another MLP; (3) with the generated real and fake attributes as inputs, generate features using the architecture in §4.1.1. All of this can be inferred automatically from the data. With this design, we can alleviate mode collapse (Figure 5, right) while preserving flexibility and privacy.

4.2 Discriminator

Our discriminator makes two key design choices: an MLP discriminator, and the decision to use *two* discriminators to improve fidelity.

MLP discriminator: A key design decision is whether to use an RNN or an MLP in the discriminator. The answer depends in part on the choice of loss function. As mentioned in §3.3, Wasserstein loss has been shown to improve the stability of training, and we find empirically that it is especially helpful for generating categorical data. However, optimizing the regularized Wasserstein loss requires the calculation of a second derivative of the loss function. At the time of writing, leading deep learning frameworks (TensorFlow and PyTorch) did not include tools for computing this second derivative; as such, any solutions that rely on such functionality would be less likely to gain traction. For this reason, we decided to use an MLP discriminator.

However, we find that when the average length of R^i is long, the data fidelity is low, as anticipated from our results on the naive GAN architecture. This may be because an MLP discriminator that discriminates on the entire object $O^i = (A^i, R^i)$ cannot provide precise information to the generator on how to improve the sample quality when O^i is large. We therefore introduce the following novel approach.

Auxiliary discriminator for data fidelity: To make the feedback information more effective, we introduce a second discriminator to split up the problem: discriminate only on attribute A^i. The generator gets feedback from both discriminators to improve itself during training (which we make more precise in the next section). Since generating good attributes A^i is much easier than generating the entire object O^i, the generator can learn from the second discriminator's feedback to generate high-fidelity attributes. Further, with the help of the original discriminator on O^i, the generator can then learn

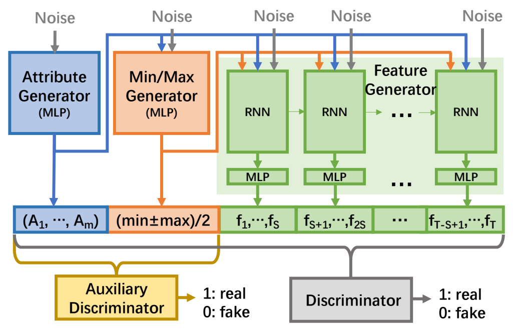

Figure 6: Architecture of DoppelGANger. The generator consists of three parts for generating attribute, min/max and features. Besides the discriminator for evaluating the entire sample, there is an auxiliary discriminator for evaluating attributes and min/max.

to generate O^i well. Empirically, we find that this architecture improves the data fidelity significantly (Appendix G).

Note that the idea of introducing a second discriminator/component in GANs is not new. But usually those new components are for extending new functions to GANs (e.g., [18, 57, 87, 99]). To the best of our knowledge, the idea of introducing a second discriminator purely for fidelity is new.

4.3 Loss function

As mentioned in §3.3, Wasserstein loss has been widely adopted for improving training stability and alleviating mode collapse. In our own empirical explorations, we find that Wasserstein loss is better than the original loss for generating categorical variables. Because categorical variables are prominent in our domain, we use Wasserstein loss.

In order to train the two discriminators simultaneously, we combine the loss functions of the two discriminators by a weighting parameter α. More specifically, the loss function is

$$\min_{G} \max_{D_1,D_2} \mathcal{L}_1(G,D_1) + \alpha \mathcal{L}_2(G,D_2) \tag{2}$$

where $\mathcal{L}_i$, $i \in \{1,2\}$ is the Wasserstein loss of the original and second discriminator, respectively: $\mathcal{L}_i = \mathbb{E}_{x \sim p_x}[T_i(D_i(x))] - \mathbb{E}_{z \sim p_z}[D_i(T_i(G(z)))] - \lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}}\left[(\nabla_{\hat{x}} D_i(T_i(\hat{x})) - 1)^2\right]$. Here $T_1(x) = x$ and $T_2(x) =$ attribute part of x. $\hat{x} := tx + (1-t)G(z)$ where $t \sim \mathrm{Unif}[0,1]$. As with all GANs, the generator and discriminators are trained alternatively until convergence. Unlike naive GAN architectures, we did not observe problems with training instability, and on our datasets, convergence required only up to 200,000 batches (400 epochs when the number of training samples is 50,000).

4.4 Putting it all together

The overall DoppelGANger architecture is in Figure 6. The data holder first trains DoppelGANger on the data they want to release. During this process, some parameters can be tuned. We provide recommendations on how to set them as follows and show the supporting results in Appendix G. After training, DoppelGANger parameters can then be released to the data consumer, who can use them to generate a desired sample size and optionally a desired attribute distribution.

Dataset	Correlated in time & attributes	Multi-dimensional features	Variable-length signals
WWT [34]	x		
MBA [20]	x	x	
GCUT [73]	x	x	x

Table 2: Challenging properties of studied datasets.

Feature batch size S (§4.1.1): This parameter controls the number of features generated at each RNN pass. Empirically, setting S so that T/S (the number of steps RNN needs to take) is around 50 gives good results, whereas prior time series GANs use $S = 1$ [11, 24]. This is how we use the data collection frequency information from §3.1.

Min/Max Generator (§4.1.3): This generator can be turned on or off. When the range of feature values varies widely across samples, it can improve data fidelity. Traditional GANs lack this generator, possibly because the kinds of data they consider are more controlled and do not need it.

Auxiliary Discriminator (§4.2): This discriminator can be turned on or off. It helps regulate data fidelity for longer, complex signals. Traditional GAN systems lack such a discriminator because they do not separate the conditional generation of attributes from features; we do this for fidelity and flexibility.

Private attribute distributions: If the data holder wants to hide an attribute distribution, DoppelGANger can retrain its Attribute Generator to a different distribution without affecting other parts of the network. This again is unique to DoppelGANger because of its isolated attribute generation.

5 Evaluation

We evaluate DoppelGANger[3] for fidelity, flexibility, and privacy on three datasets, whose properties are summarized in Table 2. These datasets are chosen to exhibit different combinations of the challenges in §1-3. In particular, they exhibit correlations within time series and across attributes, and/or multi-dimensional features and variable feature lengths. Here we outline these datasets and our baseline algorithms. All dataset schema are included in Appendix A, and more implementation details are in Appendix B.

Wikipedia Web Traffic (WWT): This dataset tracks the number of daily views of various Wikipedia articles, starting from July 1st, 2015 to December 31st, 2016.[4] In our language, each *object* is a page view counter for one Wikipedia page.

[3] The code is available at https://github.com/fjxmlzn/DoppelGANger
[4] https://www.kaggle.com/c/web-traffic-time-series-forecasting

Each object has three *attributes*: a Wikipedia domain, type of access (e.g., mobile, desktop), and type of agent (e.g., spider). Each object has one feature: the number of daily page views.
Measuring Broadband America (MBA): This dataset was collected by United States Federal Communications Commission (FCC) [20] and consists of several measurements such as round-trip times and packet loss rates from several clients in geographically diverse homes to different servers using different protocols (e.g. DNS, HTTP, PING). Each *object* consists of measurements from one measurement device. Each object has three *attributes*: Internet connection technology, ISP, and US state. A record contains UDP ping loss rate (minimum across some predefined servers) and total traffic (bytes sent and received), reflecting client's aggregate Internet usage.
Google Cluster Usage Traces (GCUT): This dataset [73] contains usage traces of a Google Cluster of 12.5k machines over a period of 29 days in May 2011. We focus on the task resource usage logs, containing measurements of task resource usage, and the exit code of each task. Once the task starts running, every second the system measures its resource usage (e.g. CPU usage, memory usage), and every 5 minutes the system logs the aggregated statistics of the measurements (e.g. mean, maximum). Those resource usage values are the *features*. When the task ends, its end event type (e.g. FAIL, FINISH, KILL) is also logged. Each task has one end event type, which we treat as an *attribute*.

5.0.1 Baselines

We compare DoppelGANger to a number of representative baselines (§2.2). We discuss the specific configurations and extensions we implement in each case:
Hidden Markov models (HMM) (§2.2): While HMMs have been used for generating time series data, there is no natural way to jointly generate attributes and time series in HMM. Hence, we infer a separate multinomial distribution for the attributes. During generation, attributes are randomly drawn from the multinomial distribution on training data, independently of the time series. We use the same technique discussed in 4.1.1 to generate variable-length time series.
Nonlinear auto-regressive (AR) (§2.2): Traditional AR models can only learn to generate features. In order to jointly learn to generate attributes and features, we design the following more advanced version of AR: we learn a function f such that $R_t = f(A, R_{t-1}, R_{t-2}, ..., R_{t-p})$. To boost the accuracy of this baseline, we use a multi-layer perceptron version of f. During generation, A is randomly drawn from the multinomial distribution on training data, and the first record R_1 is drawn a Gaussian distribution learned from training data. We use the same technique as discussed in 4.1.1 to generate variable-length time series.
Recurrent neural networks (RNN) (§2.2): In this model, we train an RNN via teacher forcing [91] by feeding in the true time series at every time step and predicting the value of the time series at the next time step. Once trained, the RNN can be used to generate the time series by using its predicted output as the input for the next time step. Again, the traditional RNN can only learn to generate features. We design an advanced version where RNN takes attribute A as an additional input. During generation, A is randomly drawn from the multinomial distribution on training data, and the first record R_1 is drawn a Gaussian distribution learned from training data. We use the same technique as discussed in 4.1.1 to generate variable-length time series.
Naive GAN (§3.3): We include the naive GAN architecture (MLP generator and discriminator) in all our evaluations. We use the same padding technique as discussed in 4.1.1 to generate variable-length time series. The generated time series after the first presence of $p_1 < p_2$ will be discarded.

5.1 Fidelity

In line with prior recommendations [64], we explore how DoppelGANger captures structural data properties like temporal correlations and attribute-feature joint distributions.[5]
Temporal correlations: To show how DoppelGANger captures temporal correlations, Figure 1 shows the average autocorrelation for the WWT dataset for real and synthetic datasets (discussed in §2.2). As mentioned before, the real data exhibits a short-term weekly correlation and a long-term annual correlation. DoppelGANger captures both, as evidenced by the periodic weekly spikes and the local peak at roughly the 1-year mark, unlike our baseline approaches. It also exhibits a 95.8% lower mean square error from the true data autocorrelation than the closest baseline (Naive GAN).

The fact that DoppelGANger captures these correlations is surprising, particularly since we are using an RNN generator. Typically, RNNs are able to reliably generate time series around 20 samples, while the WWT dataset has over 500 samples. We believe this is due to a combination of adversarial training (not typically used for RNN training) and our batched sample generation. Empirically, eliminating either feature hurts the learned autocorrelation (Appendix G).

Another aspect of learning temporal correlations is generating time series of the right length. Figure 7 shows the duration of tasks in the GCUT dataset for real and synthetic datasets generated by DoppelGANger and RNN. DoppelGANger's length distribution fits the real data well, capturing the bimodal pattern in real data, whereas RNN fails. Other baselines are even worse at capturing the length distribution (Appendix C). We observe this regularly; while DoppelGANger captures multiple data modes, our baselines tend to capture one at best. This may be due to the naive randomness in the other baselines. RNNs and AR models incorporate too little randomness, causing them to learn simplified duration distributions;

[5] Such properties are sometimes ignored in the ML literature in favor of downstream performance metrics; however, in systems and networking, we argue such microbenchmarks are important.

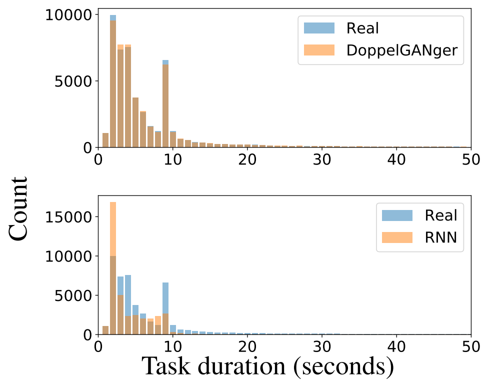

Figure 7: Histogram of task duration for the Google Cluster Usage Traces. RNN-generated data misses the second mode, but DoppelGANger captures it.

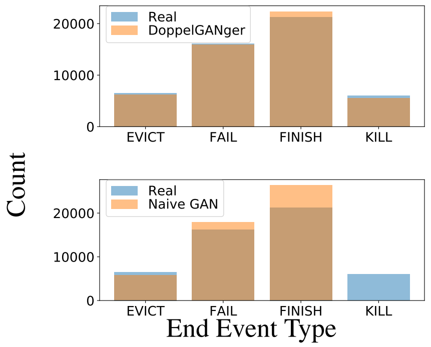

Figure 8: Histograms of end event types from GCUT.

HMMs instead are too random: they maintain too little state to generate meaningful results.

Feature-attribute correlations: Learning correct attribute distributions is necessary for learning feature-attribute correlations. As mentioned in §5.0.1, for our HMM, AR, and RNN baselines, attributes are randomly drawn from the multinomial distribution on training data because there is no clear way to jointly generate attributes and features. Hence, they trivially learn a perfect attribute distribution. Figure 8 shows that DoppelGANger is also able to mimic the real distribution of end event type distribution in GCUT dataset, while naive GANs miss a category entirely; this appears to be due to mode collapse, which we mitigate with our second discriminator. Results on other datasets are in Appendix C.

Although our HMM, AR, and RNN baselines learn perfect attribute distributions, it is substantially more challenging to learn the joint attribute-feature distribution. To illustrate this, we compute the CDF of total bandwidth for DSL and cable users in MBA dataset. Table 3 shows the Wasserstein-1 distance between the generated CDFs and the ground truth,[6] showing that DoppelGANger is closest to the real distribution. To make sense of this result, Figures 9(a) and 9(b) plot the full CDFs. Most of the baselines capture the fact that cable users consume more bandwidth than DSL users. However, DoppelGANger appears to excel in regions of the distribution with less data, e.g., very small bandwidth levels.

[6] Wasserstein-1 distance is the integrated absolute error between 2 CDFs.

	DoppelGANger	AR	RNN	HMM	Naive GAN
DSL	**0.68**	1.34	2.33	3.46	1.14
Cable	**0.74**	6.57	2.46	7.98	0.87

Table 3: Wasserstein-1 distance of total bandwidth distribution of DSL and cable users. Lower is better.

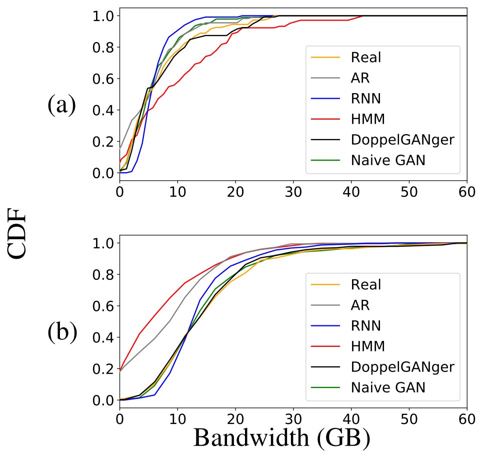

Figure 9: Total bandwidth usage in 2 weeks in MBA dataset for (a) DSL and (b) cable users.

DoppelGANger does not just memorize: A common concern with GANs is whether they are memorizing training data [7, 71]. To evaluate this, we ran the following experiment: for a given generated DoppelGANger sample, we find its nearest samples in the training data. We consistently observe significant differences (both in square error and qualitatively) between the generated samples and the nearest neighbors. Typical samples can be found in Appendix C.

5.1.1 Downstream case studies

Predictive modeling: Given a time series of records, users may want to predict whether an event E occurs in the future, or even forecast the time series itself. For example, in the GCUT dataset, we could predict whether a particular job will complete successfully. In this use case, we want to show that models trained on generated data generalize to real data.

We first partition our dataset, as shown in Figure 10. We split our real data into two sets of equal size: a training set A and a test set A'. We then train a generative model (e.g., DoppelGANger or a baseline) on training set A. We generate datasets B and B' for training and testing. Finally, we evaluate event prediction algorithms by training a predictor on A and/or B, and testing on A' and/or B'. This allows us to com-

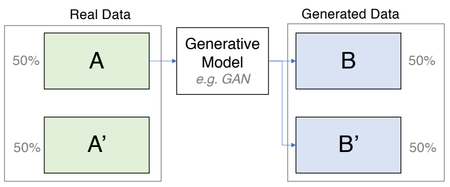

Figure 10: Evaluation setup. Our real data consists of $\{A \cup A'\}$; using training data A, we generate a set of samples $B \cup B'$. Subsequent experiments train models of downstream tasks on A or B, our training sets, and then test on A' or B'.

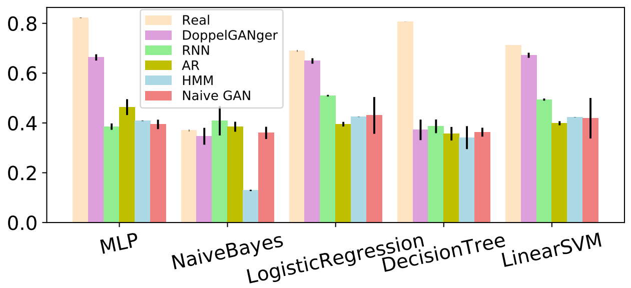

Figure 11: End event type prediction accuracy on the GCUT.

pare the generalization abilities of the prediction algorithms both within a class of data (real/generated), and generalization across classes (train on generated, test on real) [24].

We first predict the task end event type on GCUT data (e.g., EVICT, KILL) from time series observations. Such a predictor may be useful for cluster resource allocators. This prediction task reflects the correlation between the time series and underlying attributes (namely, end event type). For the predictor, we trained various algorithms to demonstrate the generality of our results: multilayer perceptron (MLP), Naive Bayes, logistic regression, decision trees, and a linear SVM. Figure 11 shows the test accuracy of each predictor when trained on generated data and tested on real. Real data expectedly has the highest test accuracy. However, we find that DoppelGANger performs better than other baselines for all five classifiers. For instance, on the MLP predictive model, DoppelGANger-generated data has 43% higher accuracy than the next-best baseline (AR), and 80% of the real data accuracy. Due to space constraints, we cover experiments on the remaining datasets in Appendix D.

Algorithm comparison: We evaluate whether algorithm rankings are preserved on generated data on the GCUT dataset by training different classifiers (MLP, SVM, Naive Bayes, decision tree, and logistic regression) to do end event type classification. We also evaluate this on the WWT dataset by training different regression models (MLP, linear regression, and Kernel regression) to do time series forecasting (details in Appendix D). For this use case, users have only generated data, so we want the ordering (accuracy) of algorithms on real data to be preserved when we train and test them on generated data. In other words, for each class of generated data,

	DoppelGANger	AR	RNN	HMM	Naive GAN
GCUT	**1.00**	**1.00**	**1.00**	0.01	0.90
WWT	**0.80**	**0.80**	0.20	-0.60	-0.60

Table 4: Rank correlation of predication algorithms on GCUT and WWT dataset. Higher is better.

we train each of the predictive models on B and test on B'. This is different from Figure 11, where we trained on generated data (B) and tested on real data (A') . We compare this ranking with the ground truth ranking, in which the predictive models are trained on A and tested on A'. We then compute the Spearman's rank correlation coefficient [82], which compares how the ranking in generated data is correlated with the groundtruth ranking. Table 4 shows that DoppelGANger and AR achieve the best rank correlations. This result is misleading because AR models exhibit minimal randomness, so *all* predictors achieve the same high accuracy; the AR model achieves near-perfect rank correlation despite producing low-quality samples; this highlights the importance of considering rank correlation together with other fidelity metrics. More results (e.g., exact prediction numbers) are in Appendix D.

5.2 Flexibility

A typical task in data-driven research is to develop algorithms for handling events that are poorly-represented in the data. For example, data consumers may want to generate more failure events. DoppelGANger provides a natural mechanism for doing so. Suppose a user wants to output more data with a particular attribute vector $A = [A_1, \ldots, A_m]$ corresponding to, say, failure events in a cluster. The user can, starting with the pre-trained attribute generator MLP, re-train it with attribute samples drawn from the desired distribution. To do so, we feed the generated attribute vectors into the same discriminator from Figure 6, but feed zeros to the time series inputs. This allows us to train the MLP generator adversarially without introducing more parameters. Notice that the user does not change the time series generator, nor does she provide additional time series samples. The conditional distribution of time series given a particular attribute combination stays the same. We give an example in Appendix E. Note that simply altering the marginal attribute distribution does not always give realistic results; for example, it is possible that introducing more failure events *should* change the conditional feature distribution. To our knowledge, no data-driven generative models capture such dependencies; doing so is an interesting question for future work. However, our approach gives more variability than replicating anomalous data events.

5.3 Privacy

Data holders' privacy concerns often fit in two categories: user privacy and business secrets. The first stems from regulatory

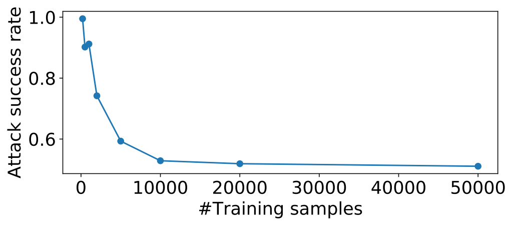

Figure 12: Membership inference attack against DoppelGANger in WWT dataset, when changing training set size.

restrictions and public relations implications, while the second affects data holders' competitive advantages. Understanding the privacy properties of GANs is an active area of research, and a full exploration is outside the scope of this (or any single) paper. We set more pragmatic goals as follows.

5.3.1 User Privacy

We focus on a narrow but common definition of user privacy in ML algorithms—the trained model should not depend too much on any individual user's data. To this end, we examine a common privacy attack (membership inference) and privacy metric (differential privacy). While we cannot conclusively say that DoppelGANger (or any GAN-based system) inherently protects user privacy, we observe two surprising findings that inform future work in this space:

1: Subsetting hurts privacy. A common practice for protecting privacy is *subsetting*, or only releasing a subset of the dataset to prevent adversaries from inferring sensitive data properties [74]. We find that subsetting actually worsens a recently-proposed class of attacks called *membership inference attacks* [40, 79]. Given a trained machine learning model and set of data samples, the goal of such an attack is to infer whether those samples were in the training data. The attacker does this by training a classifier to output whether each sample was in the training data. Notice that anonymized datasets are *trivially susceptible* to membership inference attacks.[7]

We measure DoppelGANger's vulnerability to membership inference attacks [40] on the WWT dataset. As in prior literature [40], our metric is success rate, or the percentage of successful trials in guessing whether a sample is in the training dataset. Naive random guessing gives 50%, whereas we found an attack success rate of only 51% (experimental details in Appendix F). This suggests robustness to membership inference attacks *in this case*. However, when we decrease the number of training samples, the attack success rate increases (Figure 12). For instance, with 200 training samples, the attack success rate is as high as 99.5%. We find similar trends is other datasets (Appendix F). From a machine learning point of view, this result makes sense: fewer training samples imply weaker generalization [97] and stronger overfitting. Our results suggest a practical guideline: to be more robust against membership attacks, use more training data. This contradicts the common practice of subsetting for better privacy.

[7]Since the attacker is assumed to already have the victim's data, it can trivially check if that data is in the anonymized dataset.

2: Differentially-private GANs may destroy fidelity. Differential privacy (DP) [23] has emerged as the *de facto* standard for privacy-preserving data analysis. It has been applied to deep learning [2] by clipping and adding noise to the gradient updates in stochastic gradient descent. In fact, this technique has also been used with GANs [30, 92, 93] to generate privacy-preserving time series [12, 24] to ensure that any single example in the training dataset does not disproportionately influence the model parameters. These papers argue that DP gives privacy at minimal cost to utility. To evaluate the efficacy of DP in our context, we trained DoppelGANger with DP for the WWT dataset using TensorFlow Privacy [5].

Figure 13 shows the autocorrelation of the resulting time series for different values of the privacy budget, ε. Note that smaller values of ε denote more privacy; typically, $\varepsilon \approx 1$ is considered a reasonable operating point. As ε is reduced (stronger privacy guarantees), autocorrelations become progressively worse. In this figure, we show results only for the 19th epoch, as the results become only worse as training proceeds. Complete results can be found in Appendix F. These results highlight an important point: although DP seems to destroy our autocorrelation plots, *this was not always evident from downstream metrics, such as predictive accuracy.* This highlights the need to evaluate generative time series models at a qualitative *and* quantitative level; prior work has focused mainly on the latter [12, 24]. These results also suggest that current DP mechanisms for GANs require significant improvements privacy-preserving time series generation.

5.3.2 Business Secrets

In our discussions with major data holders, a primary concern about data sharing is leaking information about the types of resources available and in use at the enterprise. Many such business secrets tend to be embedded in the attributes. For instance, a dataset's location attribute could leak market information to competitors. DoppelGANger trivially allows data holders to obfuscate the attribute generator distribution to any desired distribution using the same techniques introduced in §5.2. Notice that this is actually a *stronger* privacy guarantee than differential privacy on the attribute distribution (equivalently, it corresponds to a perfect privacy level $\varepsilon = 0$)—the data holder can choose *any* distribution to mask the original.

6 Discussion

The design of DoppelGANger was the result of a non-linear trial-and-error process in applying GANs to networking and systems datasets. We conclude with some key lessons.

Domain insight is critical: Our design process highlighted a number of fairly domain-specific problems, such as mode collapse for signals with wide dynamic ranges and poor temporal

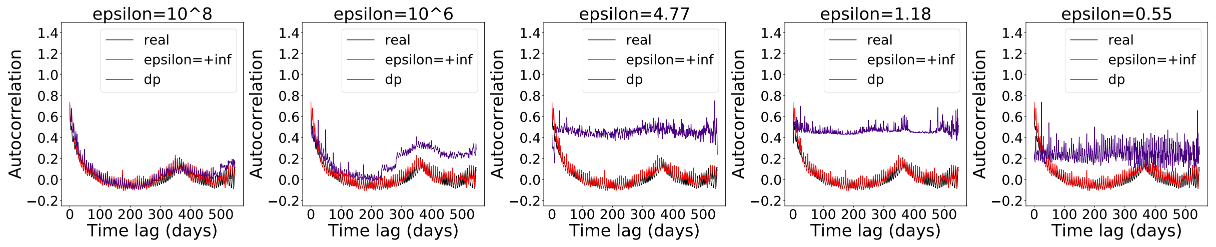

Figure 13: Autocorrelation vs time lag (in days) for real, ε = +inf and differential privacy (dp) with different values of ε).

correlations without batching. These problems do not arise in prior work, possibly due to constrained data types in other domains [11, 24]. However, these domain-specific problems significantly affected our design, leading to components like the min/max generator and batched outputs. We expect that further insights may arise as we move on to progressively harder classes of time series, such as network traces.

Differentially-private ML is not ready for prime time: At first glance, the burgeoning literature at the intersection of privacy and machine learning/generative models appeared to offer a promising solution to the privacy problem in DoppelGANger. Unfortunately, our experiments suggest a significant gap between the "hype" surrounding these theoretical approaches and the reality. As such, we find that for many of our datasets the fidelity-privacy tradeoff is far from desirable in practice, and serves as a call for further practical research in this space.

"Less is more" for privacy leakage: From anecdotal conversations, the instinct of systems operators interested in data sharing seems to release generative models learned on small datasets, in an effort to bound their "attack surface." Perhaps counterintuitively, we find that this seemingly natural strategy can be counterproductive, making the models susceptible to simple membership inference attacks! As DoppelGANger-like systems become more mature and part of operational workflows to enable data-driven collaborations, we argue that releasing generative models learned on larger datasets serve a dual benefit of providing better fidelity *and* privacy.

Acknowledgments

This research was sponsored in part by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This work was also supported in part by by Siemens AG and by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We would like to thank Safia Rahmat, Martin Otto, and Abishek Herle for valuable discussions. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575. Specifically, it used the Bridges system, which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC).

References

[1] The internet topology data kit - 2011-04. `http://www.caida.org/data/active/internet-topology-data-kit`.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[3] Abdelnaser Adas. Traffic models in broadband networks. *IEEE Communications Magazine*, 35(7):82–89, 1997.

[4] William Aiello, Anna Gilbert, Brian Rexroad, and Vyas Sekar. Sparse approximations for high fidelity compression of network traffic data. In *Proceedings of the 5th ACM SIGCOMM conference on Internet measurement*, pages 22–22. USENIX Association, 2005.

[5] Galen Andrew, Steve Chien, and Nicolas Papernot. Tensorflow privacy. `https://github.com/tensorflow/privacy`, 2019.

[6] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[7] Sanjeev Arora and Yi Zhang. Do gans actually learn the distribution? an empirical study. *arXiv preprint arXiv:1706.08224*, 2017.

[8] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th international conference on World Wide Web*, pages 181–190. ACM, 2007.

[9] Dziugas Baltrunas, Ahmed Elmokashfi, and Amund Kvalbein. Measuring the reliability of mobile broadband networks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 45–58. ACM, 2014.

[10] Kevin Bauer, Damon McCoy, Ben Greenstein, Dirk Grunwald, and Douglas Sicker. Physical layer attacks on unlinkability in wireless lans. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 108–127. Springer, 2009.

[11] Brett K Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, and Casey S Greene. Privacy-preserving generative deep neural networks support clinical data sharing. *BioRxiv*, page 159756, 2017.

[12] Brett K Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, Ran Lee, Sanjeev P Bhavnani, James Brian Byrd, and Casey S Greene. Privacy-preserving generative deep neural networks support clinical data sharing. *Circulation: Cardiovascular Quality and Outcomes*, 12(7):e005122, 2019.

[13] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[14] Zachary S Bischof, Fabian E Bustamante, and Nick Feamster. Characterizing and improving the reliability of broadband internet access. *arXiv preprint arXiv:1709.09349*, 2017.

[15] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. Optimization of resource provisioning cost in cloud computing. *IEEE transactions on services Computing*, 5(2):164–177, 2011.

[16] Rudrasis Chakraborty, Chun-Hao Yang, Xingjian Zhen, Monami Banerjee, Derek Archer, David Vaillancourt, Vikas Singh, and Baba Vemuri. A statistical recurrent model on the manifold of symmetric positive definite matrices. In *Advances in Neural Information Processing Systems*, pages 8883–8894, 2018.

[17] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. Auto: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 191–205. ACM, 2018.

[18] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[19] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. *arXiv preprint arXiv:1703.06490*, 2017.

[20] Federal Communications Commission. Raw data - measuring broadband america - seventh report, 2018. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-seventh.

[21] John M Conroy and Dianne P O'leary. Text summarization via hidden markov models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 406–407. ACM, 2001.

[22] Bo Dai, Sanja Fidler, Raquel Urtasun, and Dahua Lin. Towards diverse and natural image descriptions via a conditional gan. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2970–2979, 2017.

[23] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[24] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[25] Behrouz Farhang-Boroujeny. *Adaptive filters: theory and applications*. John Wiley & Sons, 2013.

[26] William Fedus, Ian Goodfellow, and Andrew M Dai. Maskgan: better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*, 2018.

[27] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine learning*, 32(1):41–62, 1998.

[28] Rana Forsati and Mohammad Reza Meybodi. Effective page recommendation algorithms based on distributed learning automata and weighted association rules. *Expert Systems with Applications*, 37(2):1316–1330, 2010.

[29] Maayan Frid-Adar, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Synthetic data augmentation using gan for improved liver lesion classification. In *2018 IEEE 15th international symposium on*

biomedical imaging (ISBI 2018), pages 289–293. IEEE, 2018.

[30] Lorenzo Frigerio, Anderson Santana de Oliveira, Laurent Gomez, and Patrick Duverger. Differentially private generative adversarial networks for time series, continuous, and discrete open data. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 151–164. Springer, 2019.

[31] Song Fu and Cheng-Zhong Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, page 41. ACM, 2007.

[32] Alicia Mateo González, AM Son Roque, and Javier García-González. Modeling and forecasting electricity prices with input/output hidden markov models. *IEEE Transactions on Power Systems*, 20(1):13–24, 2005.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[34] Google. Web traffic time series forecasting, 2018. https://www.kaggle.com/c/web-traffic-time-series-forecasting.

[35] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. *ACM SIGCOMM Computer Communication Review*, 44(4):455–466, 2015.

[36] John T Guibas, Tejpal S Virdi, and Peter S Li. Synthetic medical images from dual generative adversarial networks. *arXiv preprint arXiv:1709.01872*, 2017.

[37] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[38] Changhee Han, Hideaki Hayashi, Leonardo Rundo, Ryosuke Araki, Wataru Shimoda, Shinichi Muramatsu, Yujiro Furukawa, Giancarlo Mauri, and Hideki Nakayama. Gan-based synthetic brain mr image generation. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 734–738. IEEE, 2018.

[39] Md Rafiul Hassan and Baikunth Nath. Stock market forecasting using hidden markov model: a new approach. In *Intelligent Systems Design and Applications, 2005. ISDA'05. Proceedings. 5th International Conference on*, pages 192–196. IEEE, 2005.

[40] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. Logan: Membership inference attacks against generative models. *Proceedings on Privacy Enhancing Technologies*, 2019(1):133–152, 2019.

[41] Keqiang He, Alexis Fisher, Liang Wang, Aaron Gember, Aditya Akella, and Thomas Ristenpart. Next stop, the cloud: Understanding modern web service deployment in ec2 and azure. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 177–190. ACM, 2013.

[42] Félix Hernández-Campos, Kevin Jeffay, and F Donelson Smith. Modeling and generating tcp application workloads. In *2007 Fourth International Conference on Broadband Communications, Networks and Systems (BROADNETS'07)*, pages 280–289. IEEE, 2007.

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[44] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[45] Frederick Jelinek. Markov source modeling of text generation. In *The Impact of Processing Techniques on Communications*, pages 569–591. Springer, 1985.

[46] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, et al. Via: Improving internet telephony call quality using predictive relay selection. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 286–299. ACM, 2016.

[47] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. Cfa: A practical prediction system for video qoe optimization. In *NSDI*, pages 137–150, 2016.

[48] Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljacic, and Yoshua Bengio. Gated orthogonal recurrent units: On learning to forget. *Neural computation*, 31(4):765–783, 2019.

[49] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[52] Qu Li, Hu Jianming, and Zhang Yi. A flow volumes data compression approach for traffic network based on principal component analysis. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, pages 125–130. IEEE, 2007.

[53] Tiancheng Li and Ninghui Li. On the tradeoff between privacy and utility in data publishing. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–526. ACM, 2009.

[54] Yang Li, Yu-ning Dong, Hui Zhang, Hai-tao Zhao, Haixian Shi, and Xin-xing Zhao. Spectrum usage prediction based on high-order markov model for cognitive radio networks. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 2784–2788. IEEE, 2010.

[55] Zhijing Li, Zihui Ge, Ajay Mahimkar, Jia Wang, Ben Y Zhao, Haitao Zheng, Joanne Emmons, and Laura Ogden. Predictive analysis in network function virtualization. In *Proceedings of the Internet Measurement Conference 2018*, pages 161–167. ACM, 2018.

[56] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 1505–1514, 2018.

[57] Zinan Lin, Kiran Koshy Thekumparampil, Giulia Fanti, and Sewoong Oh. Infogan-cr: Disentangling generative adversarial networks with contrastive regularizers. *arXiv preprint arXiv:1906.06034*, 2019.

[58] Bede Liu and DC Munson. Generation of a random sequence having a jointly specified marginal distribution and autocovariance. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 30(6):973–983, 1982.

[59] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *ICDCS*, pages 372–382. IEEE, 2017.

[60] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2016.

[61] Mahmoud Maqableh, Huda Karajeh, et al. Job scheduling for cloud computing using neural networks. *Communications and Network*, 6(03):191, 2014.

[62] Tony McGregor, Shane Alcock, and Daniel Karrenberg. The ripe ncc internet measurement data repository. In *International Conference on Passive and Active Network Measurement*, pages 111–120. Springer, 2010.

[63] Benjamin Melamed. An overview of tes processes and modeling methodology. In *Performance Evaluation of Computer and Communication Systems*, pages 359–393. Springer, 1993.

[64] Benjamin Melamed and Jon R Hill. Applications of the tes modeling methodology. In *Proceedings of 1993 Winter Simulation Conference-(WSC'93)*, pages 1330–1338. IEEE, 1993.

[65] Benjamin Melamed, Jon R Hill, and David Goldsman. The tes methodology: Modeling empirical stationary time series. In *Proceedings of the 24th conference on Winter simulation*, pages 135–144. ACM, 1992.

[66] Benjamin Melamed and Dimitrios E Pendarakis. Modeling full-length vbr video using markov-renewal-modulated tes models. *IEEE Journal on Selected Areas in Communications*, 16(5):600–611, 1998.

[67] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *SIGCOMM*, 2018.

[68] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.

[69] Thi Thanh Sang Nguyen, Hai Yan Lu, and Jie Lu. Web-page recommendation based on web usage and domain knowledge. *IEEE Transactions on Knowledge and Data Engineering*, 26(10):2574–2587, 2013.

[70] Antonio Nucci, Ashwin Sridharan, and Nina Taft. The problem of synthetically generating ip traffic matrices: initial recommendations. *ACM SIGCOMM Computer Communication Review*, 35(3):19–32, 2005.

[71] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.

[72] Paul Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization. *Ucla L. Rev.*, 57:1701, 2009.

[73] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, pages 1–14, 2011.

[74] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release. In *2012 IEEE Network Operations and Management Symposium*, pages 1279–1286. IEEE, 2012.

[75] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[76] Aaron Schulman, Dave Levin, and Neil Spring. On the fidelity of 802.11 packet traces. In *International Conference on Passive and Active Network Measurement*, pages 132–141. Springer, 2008.

[77] David W Scott and Stephan R Sain. Multidimensional density estimation. *Handbook of statistics*, 24:229–261, 2005.

[78] Hoo-Chang Shin, Neil A Tenenholtz, Jameson K Rogers, Christopher G Schwarz, Matthew L Senjem, Jeffrey L Gunter, Katherine P Andriole, and Mark Michalski. Medical image synthesis for data augmentation and anonymization using generative adversarial networks. In *International Workshop on Simulation and Synthesis in Medical Imaging*, pages 1–11. Springer, 2018.

[79] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[80] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 68–81. ACM, 2004.

[81] Augustin Soule, Antonio Nucci, Rene Cruz, Emilio Leonardi, and Nina Taft. How to identify and estimate the largest traffic matrix elements in a dynamic environment. In *ACM SIGMETRICS Performance Evaluation Review*, volume 32, pages 73–84. ACM, 2004.

[82] Charles Spearman. The proof and measurement of association between two things. *American journal of Psychology*, 15(1):72–101, 1904.

[83] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. In *Advances in Neural Information Processing Systems*, pages 3308–3318, 2017.

[84] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: Discovery and applications of usage patterns from web data. *Acm Sigkdd Explorations Newsletter*, 1(2):12–23, 2000.

[85] Srikanth Sundaresan, Xiaohong Deng, Yun Feng, Danny Lee, and Amogh Dhamdhere. Challenges in inferring internet congestion using throughput measurements. In *Proceedings of the 2017 Internet Measurement Conference*, pages 43–56. ACM, 2017.

[86] Latanya Sweeney. Simple demographics often identify people uniquely. *Health (San Francisco)*, 671:1–34, 2000.

[87] Kiran K Thekumparampil, Ashish Khetan, Zinan Lin, and Sewoong Oh. Robustness of conditional gans to noisy labels. In *Advances in Neural Information Processing Systems*, pages 10271–10282, 2018.

[88] Mohammad Valipour, Mohammad Ebrahim Banihabib, and Seyyed Mahmood Reza Behbahani. Comparison of the arma, arima, and the autoregressive artificial neural network models in forecasting the monthly inflow of dez dam reservoir. *Journal of hydrology*, 476:433–441, 2013.

[89] Kashi Venkatesh Vishwanath and Amin Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking (TON)*, 17(3):712–725, 2009.

[90] Ning Wang. Geant/abilene network topology data and traffic traces, 2004.

[91] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[92] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*, 2018.

[93] Chugui Xu, Ju Ren, Deyu Zhang, Yaoxue Zhang, Zhan Qin, and Kui Ren. Ganobfuscator: Mitigating information leakage under gan via differential privacy. *IEEE Transactions on Information Forensics and Security*, 14(9):2358–2371, 2019.

[94] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.

[95] Zainab R Zaidi and Brian L Mark. Mobility estimation for wireless networks based on an autoregressive model. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 6, pages 3405–3409. IEEE, 2004.

[96] G Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

[97] Pengchuan Zhang, Qiang Liu, Dengyong Zhou, Tao Xu, and Xiaodong He. On the discrimination-generalization tradeoff in gans. *arXiv preprint arXiv:1711.02771*, 2017.

[98] Yizhe Zhang, Zhe Gan, and Lawrence Carin. Generating text via adversarial training. In *NIPS workshop on Adversarial Training*, volume 21, 2016.

[99] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

A Datasets

Google Cluster Usage Trace: Due to the substantial computational requirements of training GANs and our own resource constraints, we did not use the entire dataset. Instead, we uniformly sampled a subset of 100,000 tasks and used their corresponding measurement records to form our dataset. This sample was collected after filtering out the following categories of objects:

- 197 (0.17%) tasks don't have corresponding end events (such events may end outside the data collection period)
- 1403 (1.25%) tasks have discontinuous measurement records (i.e., the end timestamp of the previous measurement record does not equal the start timestamp of next measurement record)
- 7018 (6.25%) tasks have an empty measurement record
- 3754 (3.34%) tasks have mismatched end times (the timestamp of the end event does not match the ending timestamp of the last measurement).

The maximum feature length in this dataset is 2497, however, 97.06% samples have length within 50. The schema of this dataset is in Table 5.

Wikipedia Web Traffic Dataset: The original datasets consists of 145k objects. After removing samples with missing data, 117k objects are left, from which we sample 100k objects for our evaluation. All samples have feature length 550. The schema of this dataset is in Table 6.

FCC MBA dataset: We used the latest cleaned data published by FCC MBA in December 2018 [20]. This datasets contains hourly traffic measurements from 4378 homes in September and October 2017. However, a lot of measurements are missing in this dataset. Considering period from 10/01/2017 from 10/15/2017, only 45 homes have complete network usage measurements every hour. This small sample set will make us hard to understand the actual dynamic patterns in this dataset. To increase number of valid objects, we take the average of measurements every 6 hours for each home. As long as there is at least one measurement in each 6 hours period, we regard it as a valid object. Using this way, we get 739 valid objects with measurements from 10/01/2017 from 10/15/2017, from which we sample 600 objects for our evaluation. All samples have feature length 56. The schema of this dataset is in Table 7.

B Implementation Details

DoppelGANger: Attribute generator and min/max generator are MLPs with 2 hidden layers and 100 units in each layer. Feature generator is 1 layer of LSTM with 100 units. Softmax layer is applied for categorical feature and attribute output. Sigmoid or tanh is applied for continuous feature and attribute output, depending on whether data is normalized to [0,1] or [-1,1] (this is configurable). The discriminator and auxiliary discriminator are MLP with 4 hidden layers and 200 units in each layer. Gradient penalty weight was 10.0 as suggested in [37]. The network was trained using Adam optimizer with learning rate of 0.001 and batch size of 100 for both generators and discriminators.

AR: We used $p = 3$, i.e., used the past three samples to predict the next. The AR model was an MLP with 4 hidden layers and 200 units in each layer. The MLP was trained using Adam optimizer [50] with learning rate of 0.001 and batch size of 100.

RNN: For this baseline, we used LSTM (Long short term memory) [43] variant of RNN. It is 1 layers of LSTM with 100 units. The network was trained using Adam optimizer with learning rate of 0.001 and batch size of 100.

Naive GAN: The generator and discriminator are MLPs with 4 hidden layers and 200 units in each layer. Gradient penalty weight was 10.0 as suggested in [37]. The network was trained using Adam optimizer with learning rate of 0.001 and batch size of 100 for both generator and discriminator.

C Additional Fidelity Results

Temporal length: Figure 14 shows the length distribution of DoppelGANger and baselines in GCUT dataset. It is clear that DoppelGANger has the best fidelity.

Attribute distribution: Figure 15, 16, 17 show the histograms of Wikipedia domain, access type, and agent of Naive GAN and DoppelGANger. DoppelGANger learns the distribution pretty well, whereas naive GAN cannot.

Figure 18, 19, 22 show the histograms of ISP, technology, and state of DoppelGANger and all baselines. Again, for HMM, AR, RNN baselines, the attributes are directly drawn from the empirical distribution on training data, therefore, they will have the best fidelity. We compute the Jensen–Shannon divergence (JSD) between generated distribution and the real distribution in Figure 20, 21, 23. We see that DoppelGANger's JSD is actually very close to HMM, AR, RNN. (This in part is because this dataset has only 600 samples to compare.)

DoppelGANger does not simply memorize training samples: Figure 24, 25, 26 show the some generated samples from DoppelGANger and their nearest (based on squared error) samples in training data from the three datasets. The results show that DoppelGANger is not memorizing training samples. To achieve the good fidelity results we have shown before, DoppelGANger must indeed learn the underlying structure of the samples.

D Additional Case Study Results

Predictive modeling: For the WWT dataset, the predictive modeling task involves forecasting of the page views for next 50 days, given those for the first 500 days. We want to learn a (relatively) parsimonious model that can take an arbitrary

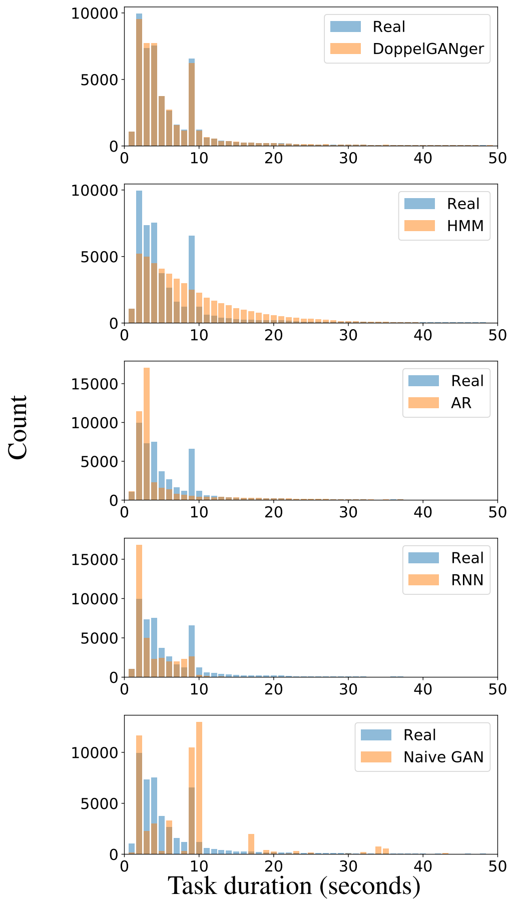
Figure 14: Histogram of task duration for the GCUT dataset. DoppelGANger gives the best fidelity.

length-500 time series as input and predict the next 50 time steps. For this purpose, we train various regression models: an MLP with five hidden layers (200 nodes each), and MLP with just one hidden layer (100 nodes), a linear regression model, and a Kernel regression model using an RBF kernel. To evaluate each model, we compute the so-called *coefficient of determination*, R^2, which captures how well a regression model describes a particular dataset.[8]

Figure 27 shows the R^2 for each of these models for each of our generative models and the real data. Here we train each regression model on generated data (B) and test it on real data (A'), hence it is to be expected that real data performs best. It is clear that DoppelGANger performs better than other baselines for all regression models. Note that sometimes RNN, AR, and naive GANs baselines have large negative R^2 which are therefore not visualized in this plot.

Algorithm comparison: Figure 28, 29 show the ranking of prediction algorithms on DoppelGANger's and baselines' generated data. Combined with 4, we see that DoppelGANger and AR are the best for preserving ranking of prediction algorithms.

[8]For a time series with points (x_i, y_i) for $i = 1, \ldots, n$ and a regression function $f(x)$, R^2 is defined as $R^2 = 1 - \frac{\sum_i (y_i - f(x_i))^2}{\sum_i (y_i - \bar{y})^2}$ where $\bar{y} = \frac{1}{n}\sum_i y_i$ is the mean y-value. Notice that $-\infty \le R^2 \le 1$, and a higher score indicates a better fit.

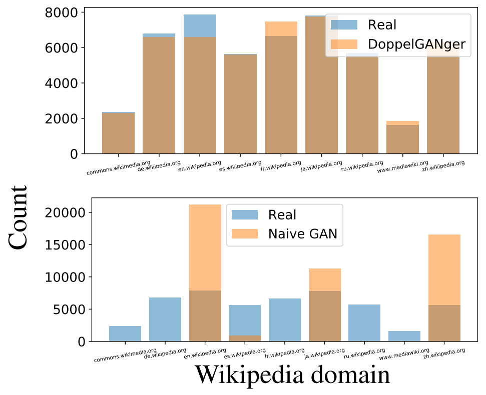
Figure 15: Histograms of Wikipedia domain from WWT dataset.

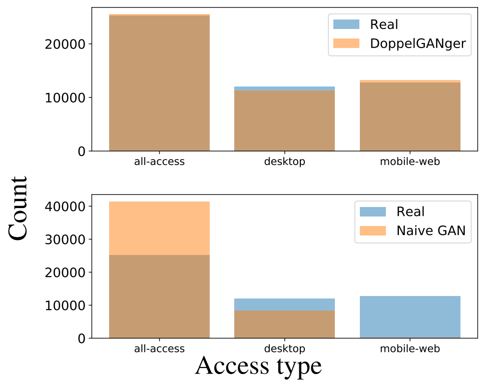
Figure 16: Histograms of access type from WWT dataset.

E Additional Flexibility Results

To illustrate the process outlined in Section 5.2, we show an example how to generate an arbitrary attribute distribution based on the Wikipedia web traffic dataset. We start with the true joint attribute distribution of domain names and access types. We then impose an arbitrary desired joint distribution (e.g., uniform, discretized Gaussian, impulse). We then re-train our attribute generator to match the target distribution. Figure 30 shows the heatmap of our target joint (Gaussian) probability distribution compared to the (very similar) distribution of the re-trained generator. This result demonstrates the ability of DoppelGANger to change attribute distribution to an arbitrary one according to the need of data holder and consumer.

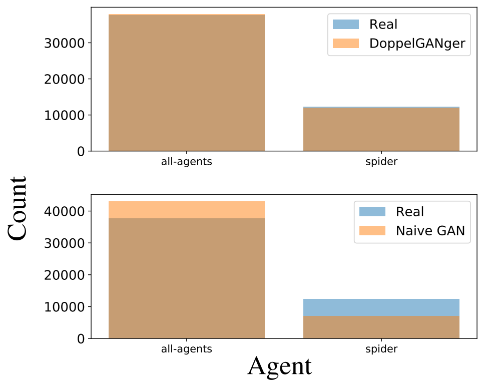

Figure 17: Histograms of agent from WWT dataset.

F Additional Privacy Results

Figure 32 shows the autocorrelation of generated page views from DoppelGANger with different differential privacy degrees (ε) and at different training epochs. We see that no matter what ε is, as the training proceeds, the fidelity of time series gets worse. This may because the noise added for differential privacy during the training process deviates the learning of GANs. On the other hand, for a fixed epoch (e.g., 19), higher ε gives poorer fidelity, which we have highlighted in the main text §5.3.1.

G Additional Design Validation Results

Feature batch size S: One parameter in DoppelGANger is feature batch size S (§4.1.1). In this section we explore how S influences the results, and thus support the recommendation we give in §4.4. We enumerate $S = 1, 5, 10, 25, 50$ on WWT dataset. Recall that in this dataset the feature (daily page view) has a weekly and annual correlation pattern. We find that when $10 \le S \le 25$, DoppelGANger stably captures both patterns during the training process. The full correlation plot is in Figure 33.

Auxiliary discriminator: Figure 34, 35 show the generated (max$\pm$min)/2 distribution from DoppelGANger with and without the auxiliary discriminator on WWT dataset. After adding the auxiliary discriminator, the distributions are learned much better.

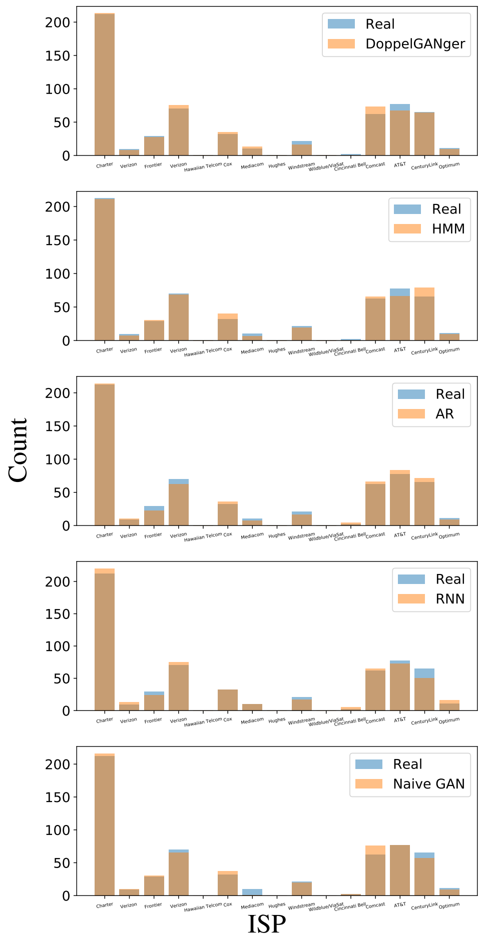

Figure 18: Histograms of ISP from MBA dataset.

Attributes	Description	Possible Values
end event type	The reason that the task finishes	FAIL, KILL, EVICT, etc.
Features	**Description**	**Possible Values**
CPU rate	Mean CPU rate	float numbers
maximum CPU rate	Maximum CPU rate	float numbers
sampled CPU usage	The CPU rate sampled uniformly on all 1 second measurements	float numbers
canonical memory usage	Canonical memory usage measurement	float numbers
assigned memory usage	Memory assigned to the container	float numbers
maximum memory usage	Maximum canonical memory usage	float numbers
unmapped page cache	Linux page cache that was not mapped into any userspace process	float numbers
total page cache	Total Linux page cache	float numbers
local disk space usage	Runtime local disk capacity usage	float numbers
Timestamp Discription		**Possible Values**
The timestamp that the measurement was conducted on. Different task may have different number of measurement records (i.e. T^i may be different)		2011-05-01 01:01, etc.

Table 5: Schema of GCUT dataset. Attributes and features are described in more detail in [73].

Attributes	Description	Possible Values
Wikipedia domain	The main domain name of the Wikipedia page	zh.wikipedia.org, commons.wikimedia.org, etc.
access type	The access method	mobile-web, desktop, all-access, etc.
agent	The agent type	spider, all-agent, etc.
Features	**Description**	**Possible Values**
views	The number of views	integers
Timestamp Discription		**Possible Values**
The date that the page view is counted on		2015-07-01, etc.

Table 6: Schema of WWT dataset

Attributes	Description	Possible Values
technology	The connection technology of the unit	cable, fiber, etc.
ISP	Internet service provider of the unit	AT&T, Verizon, etc.
state	The state where the unit is located	PA, CA, etc.
Features	**Description**	**Possible Values**
ping loss rate	UDP ping loss rate to the server that has lowest loss rate within the hour	float numbers
traffic byte counter	Total number of bytes sent and received in the hour (excluding the traffic due to the activate measurements)	integers
Timestamp Discription		**Possible Values**
The time of the measurement hour		2015-09-01 1:00, etc.

Table 7: Schema of MBA dataset

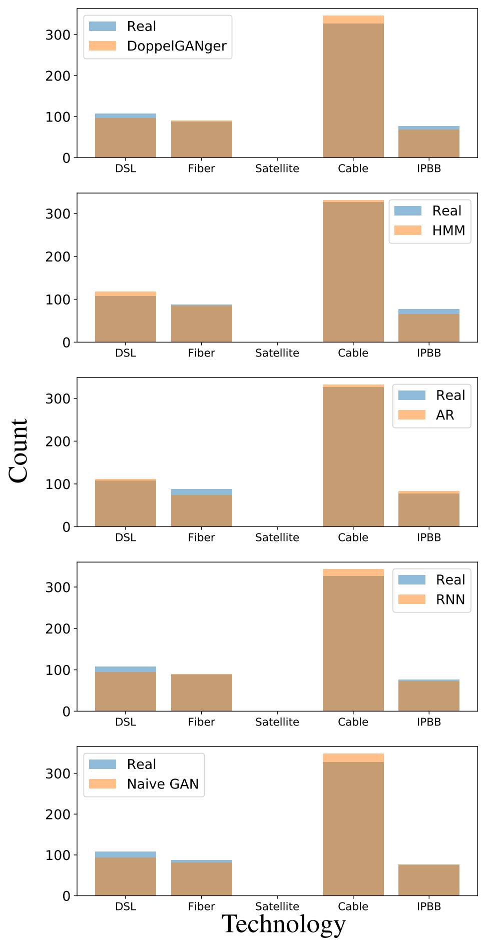

Figure 19: Histograms of technology from MBA dataset.

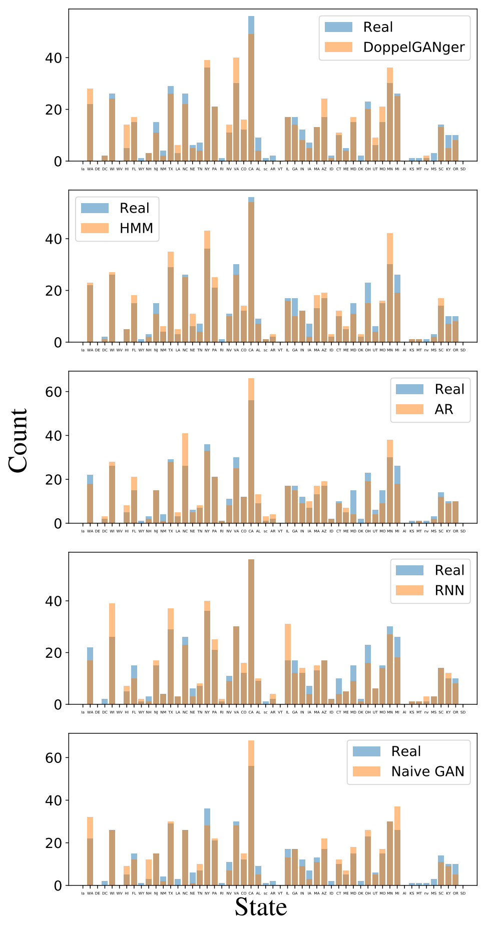

Figure 22: Histograms of state from MBA dataset.

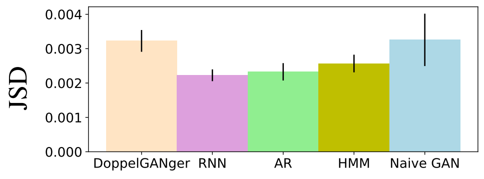

Figure 20: JSD between generated ISP distribution and real ISP distribution from MBA dataset.

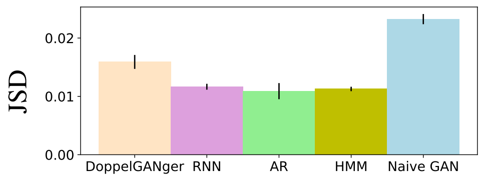

Figure 23: JSD between generated state distribution and real state distribution from MBA dataset.

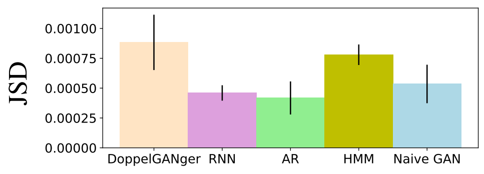

Figure 21: JSD between generated technology distribution and real technology distribution from MBA dataset.

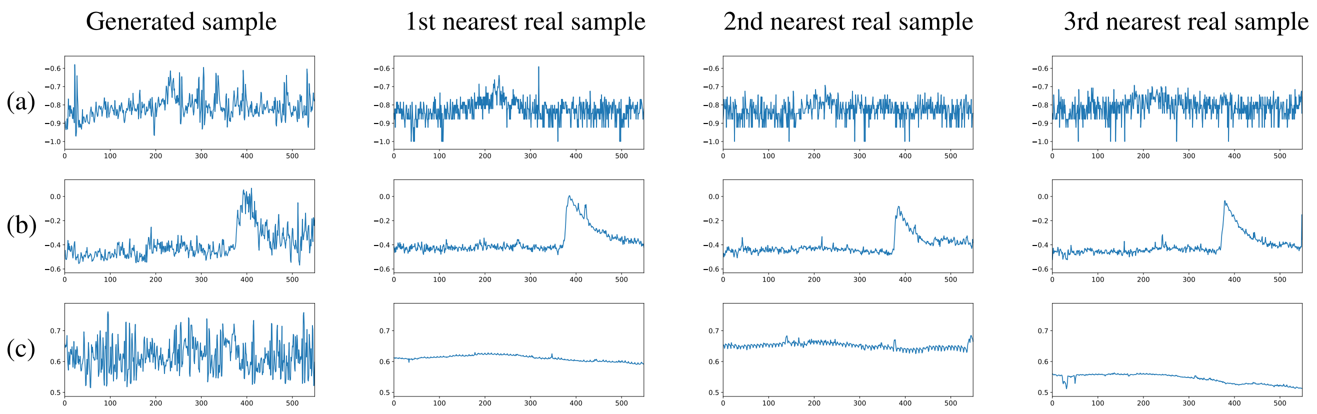

Figure 24: Three time series samples selected uniformly at random from the synthetic dataset generated using DoppelGANger and the corresponding top-3 nearest neighbours (based on square error) from the real WWT dataset. The time series shown here is daily page views (normalized).

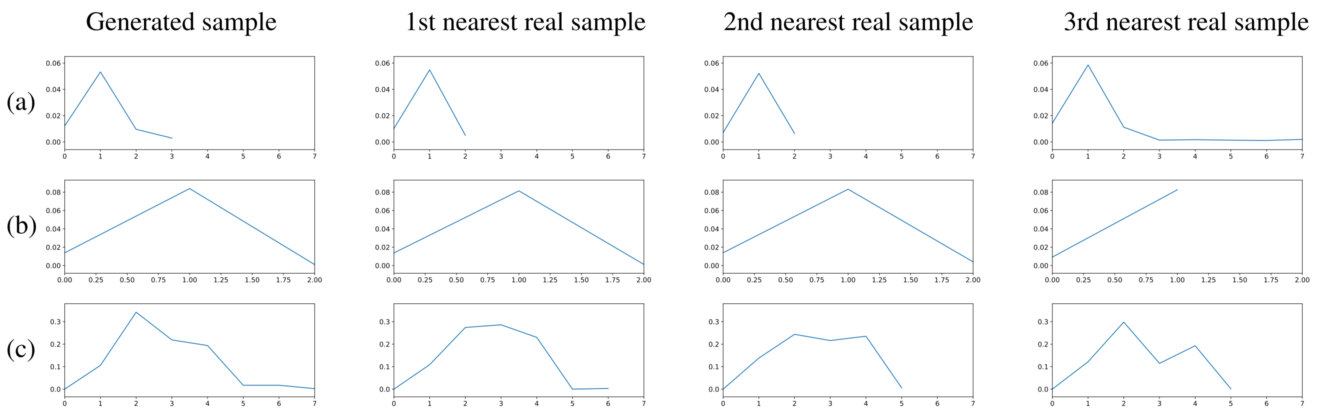

Figure 25: Three time series samples selected uniformly at random from the synthetic dataset generated using DoppelGANger and the corresponding top-3 nearest neighbours (based on square error) from the real GCUT dataset. The time series shown here is CPU rate (normalized).

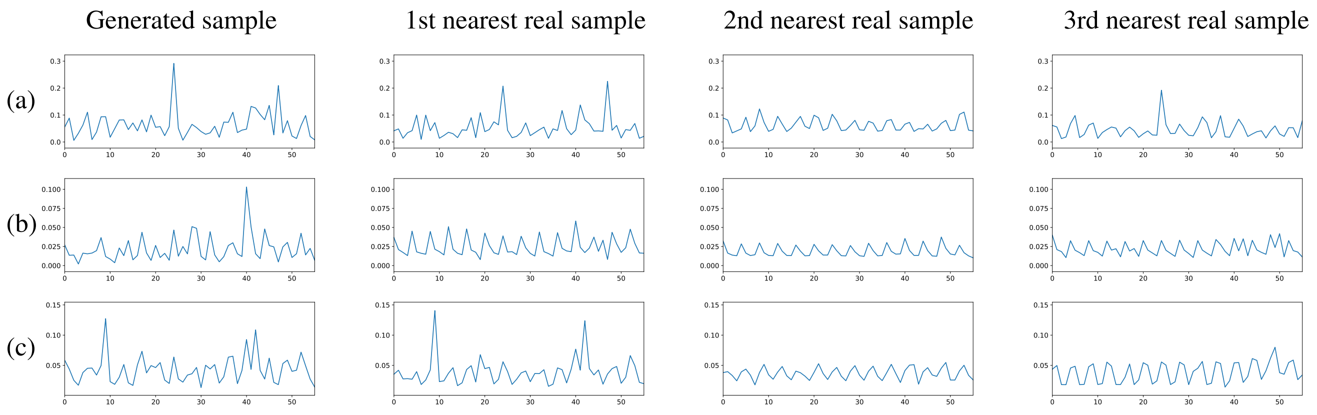

Figure 26: Three time series samples selected uniformly at random from the synthetic dataset generated using DoppelGANger and the corresponding top-3 nearest neighbours (based on square error) from the real MBA dataset. The time series shown here is traffic byte counter (normalized).

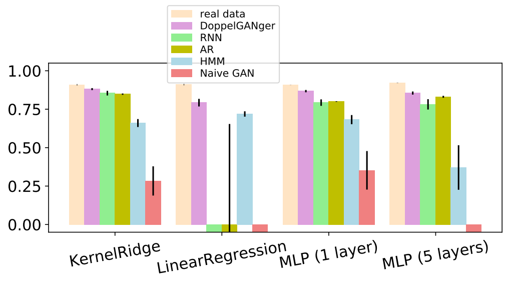

Figure 27: Coefficient of determination for WWT time series forecasting. Higher is better.

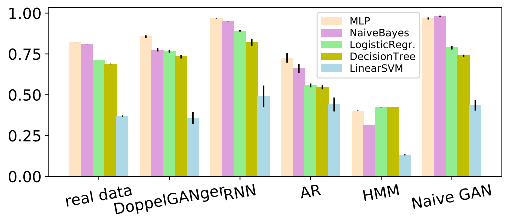

Figure 28: Ranking of end event type prediction algorithms on GCUT dataset.

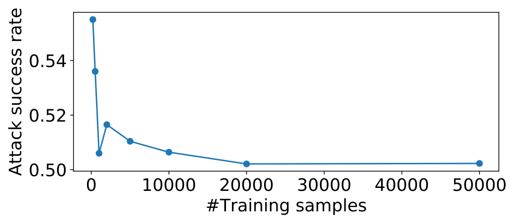

Figure 31: Success rate of membership inference attack against DoppelGANger in GCUT dataset, when changing number of training samples.

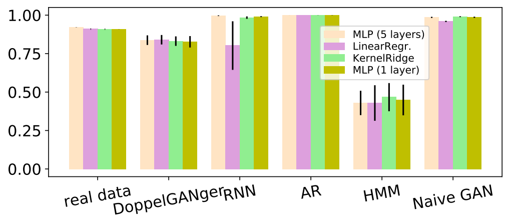

Figure 29: Ranking of traffic prediction algorithms on WWT dataset.

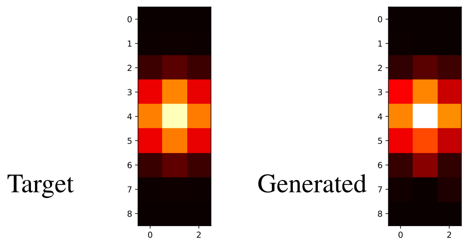

Figure 30: Target vs. generated joint distributions of attributes from the Wikipedia web traffic dataset. We impose a higher probability mass on the attribute combination corresponding to desktop traffic to domain ‘fr.wikipedia.org’.

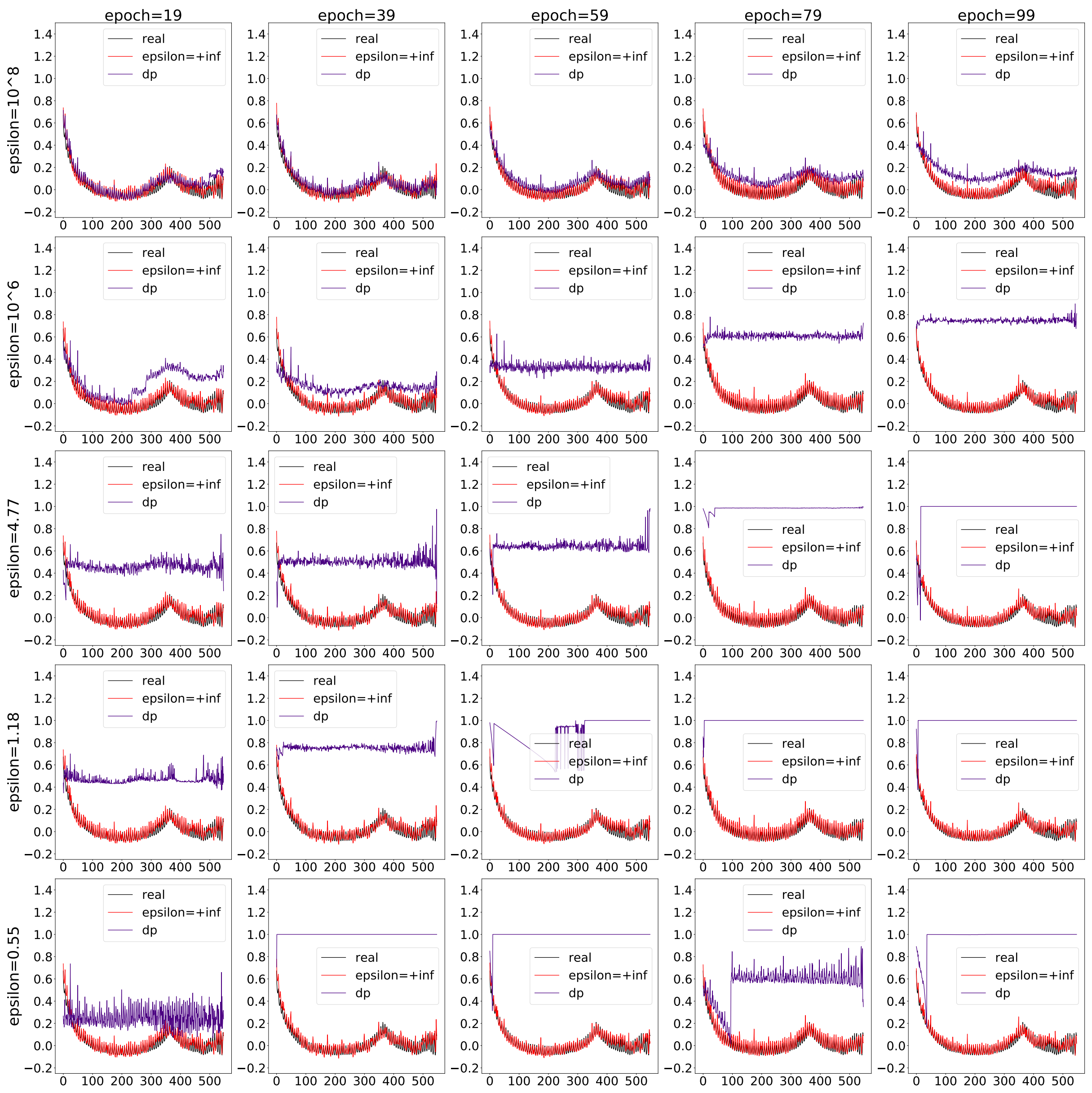

Figure 32: Autocorrelation v.s. time lag (in days) for real, ε = +inf and dp (Differential Privacy, with different values of ε) at different epochs during training.

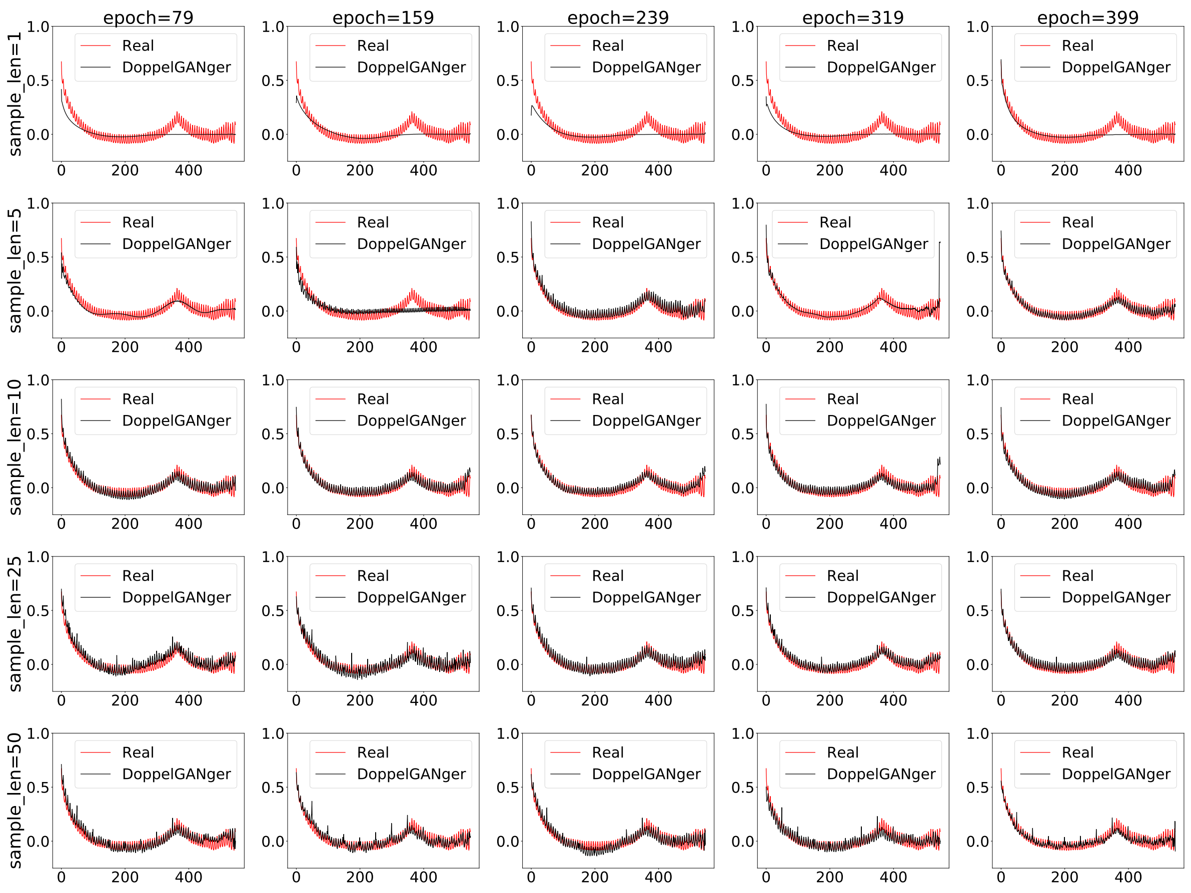

Figure 33: Autocorrelation v.s. time lag (in days) for different S at different epochs during training.

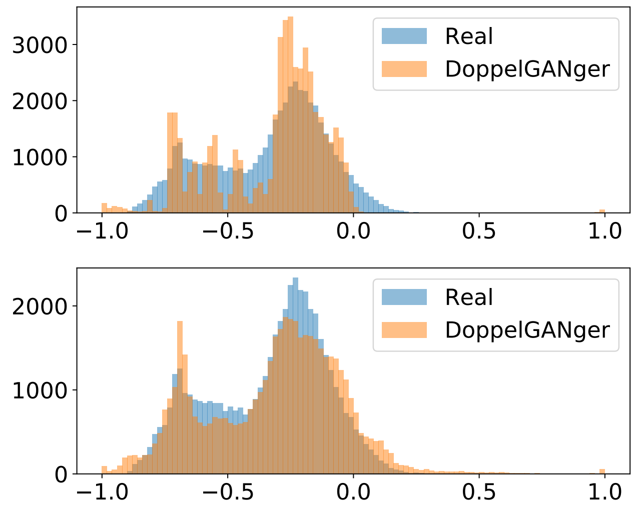

Figure 34: Distribution of (max+min)/2 from DoppelGANger (a) without and (b) with the auxiliary discriminator (WWT data).

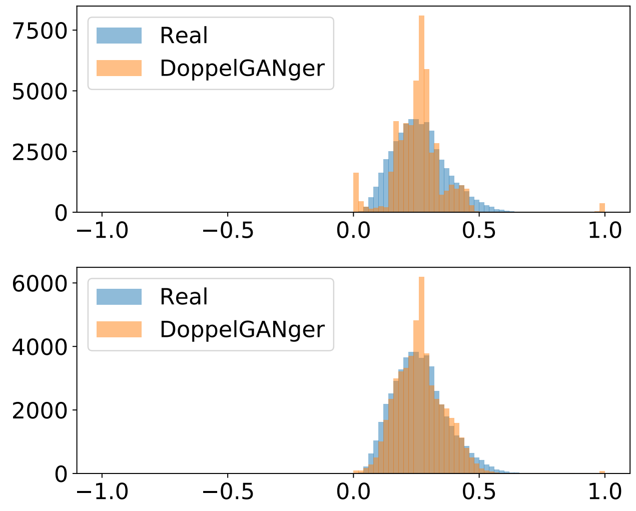

Figure 35: Distribution of (max-min)/2 from DoppelGANger (a) without and (b) with the auxiliary discriminator (WWT data).