TraceGAN: Synthesizing Appliance Power Signatures Using Generative Adversarial Networks

Alon Harell, *Student Member, IEEE,* Richard Jones, *Student Member, IEEE,*
Stephen Makonin, *Senior Member, IEEE,* and Ivan V. Bajić, *Senior Member, IEEE*

***Abstract*—Non-intrusive load monitoring (NILM) allows users and energy providers to gain insight into home appliance electricity consumption using only the building's smart meter. Most current techniques for NILM are trained using significant amounts of labeled appliances power data. The collection of such data is challenging, making data a major bottleneck in creating well generalizing NILM solutions. To help mitigate the data limitations, we present the first truly synthetic appliance power signature generator. Our solution, TraceGAN, is based on conditional, progressively growing, 1-D Wasserstein generative adversarial network (GAN). Using TraceGAN, we are able to synthesise truly random and realistic appliance power data signatures. We evaluate the samples generated by TraceGAN in a qualitative way as well as numerically by using traditional GAN evaluation methods such as the Inception score. Finally, we provide a simplistic example for the use of TraceGAN as a data augmentation tool for supervised NILM training.**

***Index Terms*—NILM, Load Disaggregation, Generative Adversarial Networks, GAN, Deep Learning, Data Synthesis, Power Signals, Smart Grid, Sustainability.**

I. Introduction

OBTAINING meaningful insight into the power consumption properties of residential users is a topic of growing importance. Such knowledge allows energy providers to better anticipate future demand, while allowing end users to identify costly appliances within their home, or other energy inefficient habits. Through a better understanding of each specific appliance's power consumption, users and providers can also begin to reduce the environmental impact of the electric grid.

Hart [1] proposed to determine the power consumption of appliances computationally through what is known as non-intrusive load monitoring (NILM). Using only the smart meter reading of a home, NILM infers the power consumption of appliances within by way of some machine learning or optimization algorithm. In most cases, algorithmically, the biggest challenge in NILM is obtaining good approximations of the distributions of appliance power consumption (p_i). More specifically, each appliance's posterior distribution conditioned on the aggregate power (p_H) measurement — $\rho\left(p_i \mid p_H\right)$. While unsupervised methods exist for this estimation, such as [2], [3], it is most commonly achieved using supervised learning methods. In a supervised setting for NILM, measurements of the aggregate and appliance specific power, taken simultaneously over significant periods of time, are used to build a model of the posterior probabilities. Some of the common models include hidden Markov models [4], [5], integer programming [6], [7], and more recently, deep neural networks [8]–[13].

Using such supervised methods means an algorithm's performance greatly depends on how well the training data represents the real distributions. In the context of NILM, this means that the training data needs to represent the true distribution of a household's power consumption characteristics. To ensure a good approximation of the real distributions, as well as a fair evaluation of performance, long term datasets must be used for training and testing.As a result, since 2011, data collection has been a main focus of NILM research, and has lead to a creation of a many publicly available datasets such as [14]–[17]. While these datasets continue to advance the development of NILM solutions, each dataset is unique (in terms of duration, sampling frequency, methodology, etc.) and may only provide a small part of the full distribution of power consumption.

When considering options for enriching NILM data, and in light of the aforementioned challenges, an alternative approach is to generate synthetic data. Our contribution is a novel approach for generating truly random appliance power signatures using generative adversarial networks (GAN). Our synthesizer, named TraceGAN, is capable of generating realistic appliance power traces in large quantities, with no hand modeling, allowing for the creation of truly random, new appliances. TraceGAN is unlike previous attempts at generating new power data [18]–[21], which are based on simple appliance modeling. TraceGAN is also novel within the existing GAN literature, as it presents an improvement over existing time-series generators based on GANs.

II. Related Work

A. Generative Adversarial Networks (GANs)

Until recently, the main use for deep neural networks (DNN) was solving problems such as classification, regression, or segmentation. While DNNs were highly successful at such tasks, including NILM [8], [11], they were not able to generate synthetic data. This changed in 2014 with the introduction of generative adversarial networks (GAN) [22]. The main novelty in GAN is that instead of one neural network trained to solve an optimization problem, two competing neural networks are trained to find the equilibrium of a game.

The two players in the GAN game are known as the *generator* and the *discriminator*. The generator tries to generate

This research was made possible through the NSERC CGS-M scholarship and NSERC Discovery Grants RGPIN-2018-06192 and RGPIN-2016-04590.

Authors are with the Computational Sustainability Lab, School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada (email:: aharell@sfu.ca, rtj4@sfu.ca, smakonin@sfu.ca, ibajic@ensc.sfu.ca).

Manuscript received June 16, 2020; revised February 13, 2021.

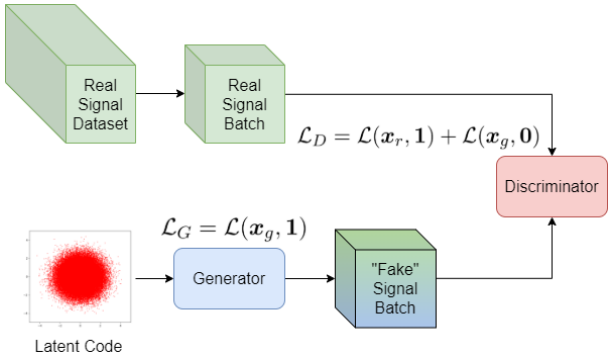

Fig. 1. GAN structure with alternating discriminator and generator training.

realistic signals from a random input known as the *latent code* (often denoted $\boldsymbol{z}$), while the discriminator attempts to successfully distinguish these generated signals from real ones. The training process is performed in turns, alternating between training the generator and the discriminator once (or more) at each turn. In terms of training loss, both the generator and discriminator are usually trained using some form of classification loss $\mathcal{L}$, often binary cross-entropy. The discriminator loss is the sum comprised of two terms: $\mathcal{L}(\boldsymbol{x_r}, \mathbf{1})$, the loss from classifying real samples, $\boldsymbol{x_r}$, to class 1; and $\mathcal{L}(\boldsymbol{x_g}, \mathbf{0})$, the loss from classifying generated samples, $\boldsymbol{x_g}$, to class 0. The generator loss, $\mathcal{L}_G(\boldsymbol{x_g}, \mathbf{1})$ is based solely on generating samples such that the discriminator classifies them to as real (class 1). Fig. 1 shows a visual explanation of the GAN framework. The equilibrium of the GAN game is achieved when the generator can create perfectly realistic signals, so that even a perfect discriminator cannot distinguish them from real ones.

The introduction of GANs allowed DNNs to generate increasingly realistic signals such as faces or scenes [23]. However, basic GANs, sometimes known as vanilla GANs, remain difficult to train. To improve both the final outcome as well as increase the stability of GAN training, many variations on the GAN framework have been published. Goodfellow *et al.* [24] suggested label smoothing, historical averaging, and minibatch discrimination. Arjovsky *et al.* [25], [26] showed that KL divergence between real and fake sample outputs of the discriminator, the commonly used loss function in GAN training, suffered from vanishing gradients, and suggested using the Wasserstein distance instead. The corresponding GANs are referred to as Wasserstein GANs (WGANs). Gulrajani *et al.* [26] presented the gradient penalty as a way to increase the stability of WGAN training. Other improvements include using a conditional generator based on class labels [27], [28], and conditioning the generator on an input signal [29] to transform the output.

Basic GANs, mentioned above, are limited in performance as well as difficult to train. This makes vanilla GANs insufficient for the challenging task of representing the true distributions of appliance level power signatures. When approaching the development of our own GAN model, we considered two specific versions of GAN – Progressively growing GAN [30], and EEG-GAN [31], both of which use the WGAN loss with gradient penalty as the underlying GAN loss.

Karras *et al.* [30] have shown that it is beneficial to train GANs in stages. At first, coarse structure is learnt by training a GAN on highly downsampled signals. After sufficient training, the next stage of the GAN is added and the signal resolution is doubled. At this stage the weights that had previously been learnt are kept and additional layers are added. On the generator side, the layers are added at the end; whereas, on the critic side they are added at the beginning.

In [31] Hartmann *et al.* present EEG-GAN, an adaption of [30] for the generation of electroencephalogram signals. The training algorithm closely resembles that of [30], with modified architectures for generating 1-D time-series data instead of images. Despite the similarity in training, the authors do present several modifications in EEG-GAN, the combination of which was novel at the time of publication. One of particular importance to TraceGAN is the weighted, one-sided gradient penalty, which is adopted by TraceGAN and expanded on in Section III-A.

B. Power Data Synthesizers

The challenges presented by the available long-term disaggregation datasets have motivated several efforts to generate synthetic data for NILM. These efforts, varying in sophistication and scope, focus on generating realistic *aggregate* signals. In contrast, the proposed TraceGAN is focused on *appliance-level traces*. Nonetheless, these power data synthesizers all employ some techniques for simulating appliance-level data before layering it to create the aggregate.

SmartSim [19] was one of the first such power data synthesizers. SmartSim's appliance level simulation is performed by matching each appliance with one of four possible energy models: ON-OFF, ON-OFF with growth/decay, stable min-max, and random range models. Reasonable parameterizations for each of these models were extracted by the authors from real instances of the specific appliances in the Smart* dataset [32]. The estimation of these values directly from real data, taken from the Smart* dataset, inherently limits SmartSim's ability to capture the variability of real appliances. Furthermore, by copying these parameters from real data, SmartSIM provides no new appliance-level traces.

The Automated Model Builder for Appliance Loads (AMBAL) [18] and its recent iteration, ANTgen [21], approach appliance models similarly. They employ the same four general appliance classes with the addition of compound model types. Compound models are combinations of the four basic models, and are generally a better fit to real-world appliances. Model parameters are determined using the ECO [17] and Tracebase datasets [16] where active segments of each appliance are broken up according to possible internal state changes. Rather than deciding *a priori* the model class for a particular appliance, AMBAL/ANTgen selects the model fit that minimizes the mean absolute percentage error.

SynD [20] is a similar effort that instead categorizes appliances as either autonomous or user-operated. Autonomous appliances include constantly-on loads (such as a router) or appliances that are cyclic in their operation patterns (such as a fridge). User-operated appliances can involve single-pattern operation (such as a kettle) or multi-pattern operation (such as

This article has been accepted for publication in IEEE Transactions on Smart Grid. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSG.2021.3078695

a dishwasher or programmable oven). On the appliance level, power traces for SynD were measured directly by the authors and stored as templates.

The extraction of appliance models directly from real data restricts the ability of these generators to provide truly novel appliance-level traces. However, the aim of these generators is to synthetically expand the space of realistic aggregate signals, which has and will continue to contribute to the NILM community. In contrast, our work focuses on appliance-level modeling, moving past the parameterization of pre-specified appliance models, and instead making use of the rapidly developing generative-adversarial framework to elucidate entire distributions over appliance behaviour. Note that we do not compare with SHED [33], which uses similar methods, because it is designed for commercial buildings rather than residential ones.

It is also important to note that GANs have been used for NILM in [12], [13], [34]. In [34] a pretrained GAN generator is used to replace the decoder side of a denoising autoencoder based disaggregator. In [12], [13], GANs were heavily conditioned on aggregate data and simply used as a refinement method for supervised disaggregation using convolutional neural networks. However, none of these works use GANs for the purpose of generating new data, evaluate their models using conventional GAN metrics, or made their models publicly available, and as such are not comparable with TraceGAN.

III. Methodology

A. TraceGAN

Both progressive growing of GANs and EEG-GAN introduce novel methods of training GANs, with a variety of techniques for improved performance and reliable convergence. However, neither of the two methods takes advantage of class labels. Inspired by [27], [28], we extend EEG-GAN by conditioning both the generator and the critic on the specific appliance label. We name our framework TraceGAN — a conditional, progressively growing, one dimensional WGAN for generating appliance-level power traces.

The basic architecture of TraceGAN is similar to the EEG-GAN adaptation of [30]. TraceGAN contains six generator and critic blocks, each comprised of two convolutional layers and an upsampling, or downsampling layer respectively. The blocks, which are added progressively throughout the training procedure, correspond to different sampling frequencies such that, as we add each additional block, we increase the sampling rate by a factor of 2. Throughout the training process the input to the generator remains the latent code $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{N_z})$ (an N_z-dimensional random vector of i.i.d standard-normally distributed variables). Conversely, the real samples used as input to the critic are downsampled as necessary to match the current frequency of the generator output. We perform this downsampling using max-pooling in order to preserve shorter activations while maintaining sharp edges.

Following the process in [30], [31], we perform a fading procedure each time a new block is added. Fading prevents the newly introduced layers, which begin with randomly initialized wrights, from affecting the weights of previously trained layers, without the need to freeze them. Thus allowing those layers to fine-tune as we continue training. At each iteration of the fading procedure, we begin with a certain configuration of the model, in which all existing blocks have already been trained for a sufficient number of epochs — this is stage (a) in Fig. 2. We consider these blocks to be stable, and proceed to fade in the new block. During fading, seen in stage (b) of Fig. 2, the output of a new block of layers is scaled by a fading parameter α and added to the output of existing layers, which is scaled by $1-\alpha$. The fading parameter itself grows linearly from 0 to 1 over $EP_f = 1000$ epochs. Finally, once the fading is finished, we have a new stable configuration containing one additional block, seen in stage (c) of Fig. 2. This stable configuration is trained for an additional 1000 epochs until we repeat the process and fade in another block. All layers remain trainable throughout the process and the corresponding dimensionality discrepancies are resolved by a simple 1×1 convolutional layer. An illustration of this process is shown in Fig. 2.

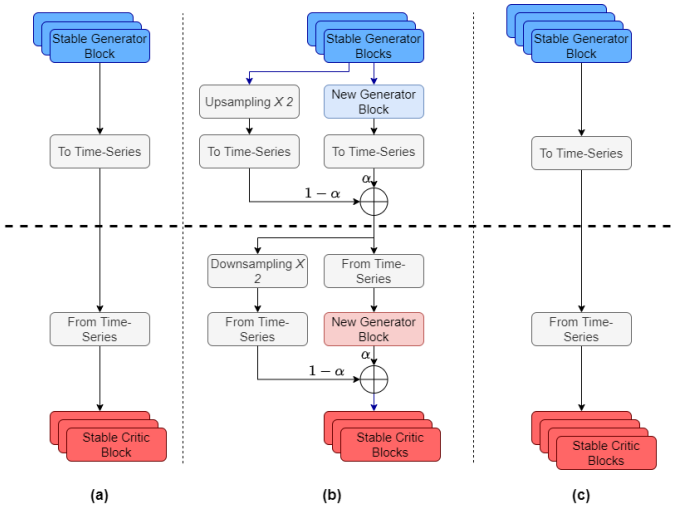

Fig. 2. The fading procedure proposed by [30] as adapted for time-series data in [31] and TraceGAN. We begin at stage (a) with a stable generator and critic, both trained for a sufficient number of epochs, during an intermediate stage of training; note that generator (critic) contains a upsampling (downsampling) step. The blocks "To Time-Series" and "From Time-Series" are implemented via 1D convolution. In (b), we see the fading stage used to introduce the next block to both the generator and the critic. On the generator side, the output of new blocks is slowly faded in, using a linearly growing parameter α, with a nearest neighbor upsampling of the output of the stable blocks. Similarly, on the critic side, the features created by the new block are slowly merged in with previous inputs to the existing critic blocks. Finally, (c) shows the blocks after the fading is complete and $\alpha = 1$. Note that the generator and critic now contain one additional stable block each. In TraceGAN, this fading is performed over 1000 epochs, allowing for knowledge obtained at earlier steps of training to slowly adapt as new layers are added.

A major novelty in TraceGAN is the introduction of conditioning, both for the generator and the critic, on the desired appliance label. Following the concepts presented in [27], we choose to condition our GAN on the input labels by including the class label as an input to both the critic and the generator. On the generator side this is done by replacing the latent code

input with $Z \in \mathbb{R}^{N_z \times C} = [\boldsymbol{z}_0^T, \boldsymbol{z}_1^T, ..., \boldsymbol{z}_C^T]$ such that:

$$\boldsymbol{z}_i^T = \begin{cases} \boldsymbol{z}^T & i = l \\ \mathbf{0}^T & \text{otherwise} \end{cases} \tag{1}$$

where N_z is the latent space dimension, $\boldsymbol{z} \in \mathbb{R}^{N_z}$ is the latent code, C is the number of different labels in the dataset, and l is the current label. In practice, this is performed by extending both the latent code and the one-hot labels to $\mathbb{R}^{N_z \times C}$ and multiplying the resulting tensors. To accommodate for the added capacity required by the conditional generator, we increase the amount of features in the input stage by a factor of C compared with the rest of the network. On the critic side, we simply extend the one-hot labels to $\mathbb{R}^{N_s \times C}$, where N_s is the current signal length, and concatenate the resulting tensor to the input signal, as illustrated in Fig. 3.

The benefits of conditioning in GANs are numerous, and are discussed at length in the original papers [27], [28]. One significant such benefit is that conditioning allows the user to train a single model to perform the generation of all the desired classes. Without conditioning, one would be required to choose between training a model on all appliances without the ability to control the class of the generated samples, or training a model individually for each appliance class. The former case is trivially inferior, since the very purpose of TraceGAN is to generate realistic appliance-specific power traces. The latter can no doubt be accomplished, but requires additional training, computation, and memory resources to come up with a separate model for each class. It is also far less elegant than having a single model that could generate all desired classes.

In TraceGAN, we also adopt many of the smaller, nuanced, practices proposed in [30], [31]. As suggested in [30], to alleviate growing magnitude issues, we strictly normalize each time-step in each feature map to have an average magnitude of 1. To improve convergence during training, we employ online weight scaling (instead of careful weight initialization). To increase the variation of generated signals, we use a simplified version of minibatch discrimination, as proposed in [30] and modified in [31], wherein the standard deviation is used as an additional feature for the final layer of the critic. The minibatch standard deviation is calculated first at each feature, at each time-step, and then averaged across both features and time to give one single value for the entire batch.

Furthermore, we use the weighted one-sided variation of the gradient penalty, as proposed in [31], and modify it to accommodate the conditional critic and generator. The gradient penalty's importance, as noted in [31], depends on the current value of the Wasserstein distance

$$D_W = \mathbb{E}_{x_g}[D_\alpha(x_g, l)] - \mathbb{E}_{x_r}[D_\alpha(x_r, l)] \tag{2}$$

where D_α is the critic output corresponding to the current fading parameter α, x_g are the generated samples, x_r are real samples, and l is the appliance class label. When D_W is large, it is important to ensure that the cause isn't the loss of the 1-Lipschitz constraint. However, when the D_W is low, it is worthwhile to focus on optimizing it directly, and assign a lower weight to the gradient penalty. In practice, this is achieved by giving an adaptive weight to the gradient penalty equal to the current D_W. It is important to note that this weight is treated as a constant for gradient purposes, to avoid undesirable gradients. The gradient penalty itself is one-sided, meaning it allows for the critic to have a smaller than 1-Lipschitiz constraint, as was considered but ultimately not chosen in [26]. In this form the gradient penalty becomes:

$$\mathcal{L}_{GP} = \lambda \cdot \max(0, D_W) \cdot \mathbb{E}_{\tilde{x} \sim P_{\tilde{x}}}\left[\max\left(0, \|\nabla_{\tilde{x}} D_\alpha(\tilde{x}, l)\|_2 - 1\right)^2\right] \tag{3}$$

where D_W is taken from Eq. 2, $\lambda = 10$ is the weight coefficient for the gradient penalty, and $\tilde{x}$ is a randomly weighted mixture of pairs of real and generated samples, each with the same label l. Recall that D_W here is treated as a constant for back-propagation purposes.

Finally, we use a small loss component to center critic output values around zero, also introduced in EEG-GAN [31]:

$$\mathcal{L}_C = \epsilon \cdot \left(\mathbb{E}_{x_r}[D_\alpha(x_r)] + \mathbb{E}_{x_g}[D_\alpha(x_g)]\right) \tag{4}$$

where $\epsilon \ll 1$, and x_r, x_g are real and generated samples, respectively. This loss helps with numerical stability as well as interpretation of the loss value during training. Combining all of the above, the final loss functions of the critic ($\mathcal{L}_D$) and the generator ($\mathcal{L}_G$) in TraceGAN are:

$$\mathcal{L}_D = \mathbb{E}_{x_g}[D_\alpha(x_g, l)] - \mathbb{E}_{x_r}[D_\alpha(x_r, l)] + \mathcal{L}_{GP} + \mathcal{L}_C \tag{5}$$

$$\mathcal{L}_G = -\mathbb{E}_{x_g}[D_\alpha(x_g, l)] \tag{6}$$

Another important difference between TraceGAN and [31] is in the method of resampling the signals. In [31], after comparing various methods, the authors use strided convolutions for downsampling in the critic, average pooling for downsampling the input data, and either linear or cubic interpolation for upsampling in the generator. We find that given the quick switching nature of appliance power traces, it is important to allow for high frequency changes in the signal, even at the price of some aliasing. For this reason we downsample the input signals using maxpooling, and perform the upsampling steps in the generator with nearest-neighbour interpolation.

B. Training

TraceGAN was trained using the REFIT [15] dataset. REFIT consists of active power consumption data (measured in watts) from 20 residential homes, at the aggregate and appliance level, sampled at 1/8 Hz. The REFIT dataset was prepared by following the prescription of some recent work to ensure consistent sampling [9]. Because not all of the 20 houses contain the same appliances, we chose appliances that were available in multiple houses. We also wanted to ensure that these appliances were relatively prevalent, were significant in their contribution to whole-house energy consumption, and that they exemplified each of the four appliance types as defined by [1] (and expanded by [10]): ON-OFF, Multi-state, Variable Load, and Always-ON (or periodic). Of the appliances available in REFIT, five that satisfied the above considerations were used: refrigerators (along with freezers, and hybrid fridge-freezers), washing machines, tumble dryers, dishwashers, and microwaves.

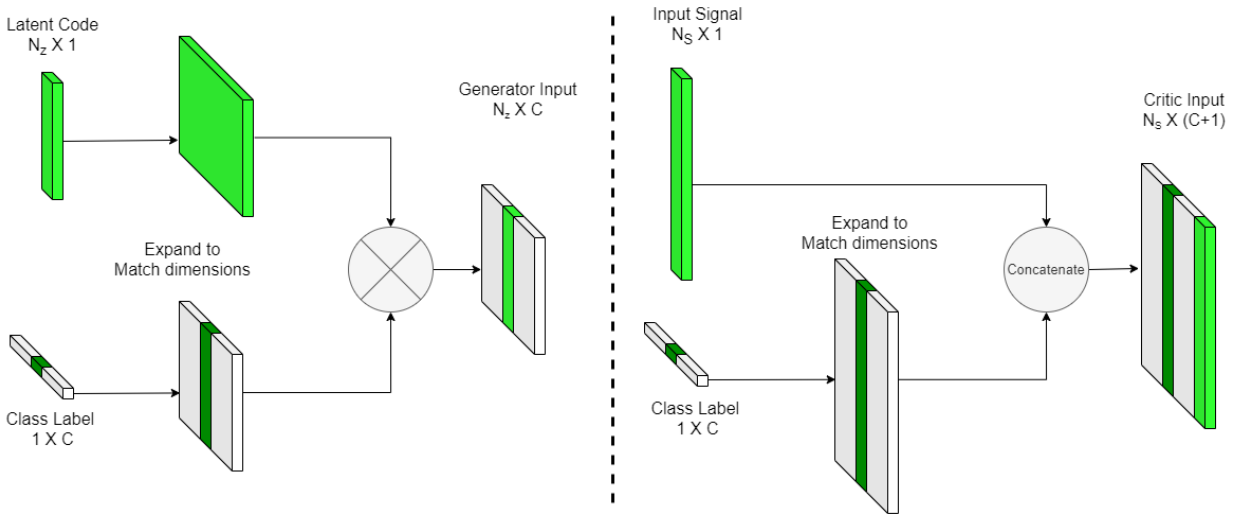

Fig. 3. TraceGAN's method of conditioning the generator and critic. On the generator side (left), the input latent code and the one-hot class label are both extended and then multiplied. Effectively, this is equivalent to placing a copy of the latent code in the corresponding column matrix which is zero everywhere else. On the critic side (right), we perform a similar extension of the class labels, but then simply concatenate the resulting tensor to the input signal.

Of course, due to the data balancing procedure described below, TraceGAN can be used to generate novel instances of all appliance in the dataset, including less prevalent ones. However, appliances with fewer unique occurrences will likely result in the reduction of the diversity of generated samples in that appliance class. Each instance of these five appliances were arranged into approximately five hour windows, centered around the available activations. We located these activations by first-order differences in power that were larger than 50 watts.

Windows were then filtered according to two conditions: First, the energy contained in the window should be appreciably larger than the "steady-state" contribution to the energy (taken here to be the sum of the window mean and half the window standard deviation). In other words, after ignoring the samples less than this value, the remaining energy contained in the window should be above some threshold, set in our work to be 33.33 Watt-hours. This condition ensures that low-energy windows, where the activation was falsely detected due to sensor noise, are excluded. This condition also filters out windows that may contain significant energy, but have little useful structural information - mainly windows composed of a constant level of power.

Secondly, we calculate the Hoyer sparsity metric [35], S, for $\boldsymbol{\delta}(w_i)$ - a vector of length n containing the discrete first-order differences in each window w_i:

$$S_{\boldsymbol{\delta}(w_i)} = \frac{\sqrt{n} - \frac{\|\boldsymbol{\delta}(w_i)\|_1}{\|\boldsymbol{\delta}(w_i)\|_2}}{\sqrt{n} - 1} \tag{7}$$

where $\|\boldsymbol{\delta}(w_i)\|_1$ and $\|\boldsymbol{\delta}(w_i)\|_2$ are the ℓ_1 and ℓ_2-norms of $\boldsymbol{\delta}(w_i)$, respectively. At its extremes, the Hoyer sparsity metric is zero when every sample in $\boldsymbol{\delta}(w_i)$ is the same (meaning the ℓ_1-norm is larger than the ℓ_2-norm by a factor of $\sqrt{n}$), and unity when there is only one non-zero sample in $\boldsymbol{\delta}(w_i)$ (i.e., highly sparse). By requiring the sparsity metric to be larger than 0.5, we ensure that windows are not overly noisy, further maximizing the structural information contained in them. The windowing procedure was further constrained by requiring the separation between windows to exceed half of the window size, avoiding capturing the same activation many times (e.g., in the case of appliance activations with significant noise or quick switching).

The remaining windowed dataset was then balanced by forcing a desired number of activation windows for each instance of each appliance (a house can have zero, one, or more instances of each appliance type). In the case of over-representation, a random subset of 300 windows was selected, of which a random 100 windows contributed to each training epoch. In the case of under-representation, those appliance instances with fewer than 100 activation windows (but more than 5) had their windows randomly repeated such that there were a total of 100 windows for each appliance instance. Finally, to ensure equal representation in the training set, the number of appliance instances was itself balanced by randomly repeating under-represented appliances. All windows of each given appliance were then shifted and scaled according to the overall mean and standard deviation of the entire dataset.

Finally, before every epoch, windows were shifted randomly in time to avoid biasing the network towards specific activation locations within each window. The shifted windows were then downsampled to match the resolution of the current training stage. We utilized the Adam [36] optimizer for training TraceGAN, setting $lr = 0.001$ and $\beta = (0, 0.99)$ We trained each stage of TraceGAN for 2000 epochs, out of which the first 1000 included fading with linearly changing weights. See Algorithm 1 for full details.

IV. Experiments

We present both a qualitative analysis of the TraceGAN-generated power traces as well as their quantitative evaluation, based on adaptations of commonly used GAN evaluation methods to 1-D power traces. We compare quantitative metrics with two other appliance power trace synthesizers: SynD [20], and ANTgen [21], which is a more up-to-date version of AMBAL. SmartSim [19] is not included in the comparison

because the published sample data is of insufficient size for accurate comparison with other methods in these experiments.

When generating signals using TraceGAN, we found it beneficial to add two simple post-processing steps: we ensure that at any given time-step the generated power is larger than zero; and we discard any generated signals that do not meet the energy threshold designated for the training data (and replace them with new generated samples).

Algorithm 1 TraceGAN Training Procedure

Input: Real samples with corresponding labels $(x_R, l) \in X_R$; Conditional Generator $G(\boldsymbol{z}, l)$; Conditional Critic $D(\boldsymbol{x}, l)$; optimizers for G, D.

Parameters: N_b: Number of blocks for G, D; EP_b = 2000: number of training epochs per block; $EP_f = 1000$: number of fading epochs; R: ratio of critic to generator training iterations.

for $n = 1, 2, \ldots, N_b$ **do**
  Add Block to G, D
  for $ep = 1, 2, \ldots, EP_b$ **do**
    Set $\alpha = \min(1, ep/EP_f)$
    Set G_α, D_α according to Fig. 2
    Randomize appliance starting points and downsample X_R by $2^{N_b - n}$
    Select a minibatch of real samples and labels: $\boldsymbol{x}_R, \boldsymbol{l}$
    Generate a mini-batch of samples using labels: $\boldsymbol{x}_G = G_\alpha\big(\boldsymbol{z} \backsim \boldsymbol{N}(0, \mathbb{I}), \boldsymbol{l}\big)$
    $\mathcal{L}_D = \mathbb{E}_{x_g}[D_\alpha(x_g, l)] - \mathbb{E}_{x_r}[D_\alpha(x_r, l)] + \mathcal{L}_{GP} + \mathcal{L}_C$
    Take optimizer step for D
    if $ep == 0 \mod R$ **then**
      generate a mini-batch of samples using labels: $\boldsymbol{x}_G = G_\alpha\big(\boldsymbol{z} \backsim \boldsymbol{N}(0, \mathbb{I}), \boldsymbol{l}\big)$
      $\mathcal{L}_G = -\mathbb{E}_{x_g}[D_\alpha(x_g, l)]$
      Take optimizer step for G
    end if
  end for
end for

All expected value operations are approximated using the sample mean of the minibatch.

A. Quantitative Evaluation

Tasks such as segmentation, classification, regression, or disaggregation, are relatively easy to evaluate because they have a well-defined goal. While there are several different approaches to evaluating NILM [37], all methods utilize a well-defined ground truth, such as appliance power consumption or state. Unfortunately, no such ground truth exists when attempting to evaluate randomly generated signals. In fact, the attempt to assign a numerical value to measure the quality of a GAN framework is in itself a significant and challenging research problem [38]. To evaluate TraceGAN, we choose three commonly used GAN evaluation metrics, and adapt them to be applicable for power trace data.

Inception score (IS) [24] uses a pre-trained DNN-based classifier named Inception [39], to evaluate the quality of generated signals. To calculate IS, a batch of generated samples are classified using the pre-trained model. The output of this classifier can be seen as the probability that a sample belongs to each target class. A good generator is realistic, meaning we expect low entropy for the output of the classifier. Simultaneously, a good generator is also diverse, meaning we expect high entropy when averaging out all classifier outputs. To include both requirements in one numerical measure, [24] defines the Inception score as $IS = \exp\Big(\mathbb{E}\big[D_{KL}\big(p\left(y|\boldsymbol{x}\right) \parallel p\left(y\right)\big)\big]\Big)$, where D_{KL} is the KL divergence, y is the Inception model's predicted class, and $\boldsymbol{x}$ are the inputs being scored.

Because the IS is not an objective metric, it is common to compare the generator's score with the score obtained from real data. Because no such classifier is commonly used for power trace signals, we train our own model, using a one dimensional ResNet [40] architecture. To avoid biasing the model towards TraceGAN we also include training data from ECO [17] and Tracebase [16], as they were the foundation used for the ANTgen power traces. The real power traces, used as foundation for SynD, were not published, so they could not be included in classifier training. We then evaluate the IS in batches and present the mean and standard deviation for each generator, as well as the real data.

While IS has shown good correlation with human classification of real versus generated samples, it is not without its flaws. It is highly sensitive to noise and to scale, as well as mode collapse. For example, if a model can generate exactly one, highly realistic, sample for every class, it will achieve near perfect IS, without actually being a diverse generator. To avoid some of these pitfalls, [41] introduced the Frechet Inception Distance (FID). The FID uses the same classifier as IS, but instead of measuring probabilities directly at the output, it evaluates the distributions of features in the final embedding layer of the classifier. FID measures the Wasserstein 2-distance between the distribution of real and generated signal features, under a Gaussian assumption (which allows a closed-form solution). The FID is significantly less sensitive to mode collapse and noise, yet still struggles with models that directly copy large portions of the training set. Because FID is a proper distance, its value can serve as a more objective metric. We evaluate FID using the full set used for training our ResNet classifier, and generate an equivalent amount of data from each synthesizer.

A similar approach to FID, the sliced Wasserstein distance (SWD) [30] attempts to evaluate the difference between the distributions of real and generated signals directly. SWD uses 1-D projections to estimate the Wasserstein distance between two distributions, taking advantage of the closed form solution for the distance of such projections. In practice, the SWD is itself approximated using a finite set of random projections. It is common to evaluate SWD on some feature space, to make it more robust. For our work, we compare two possible feature sets: the classifier features used for FID, and a Laplacian "triangle" (a 1-D adaption of a Laplacian pyramid) using a 15-sample Gaussian kernel. Similarly to FID, we evaluate the SWD on the entire training set, and we use 10 iterations of 1000 random projections each, calculating the mean and

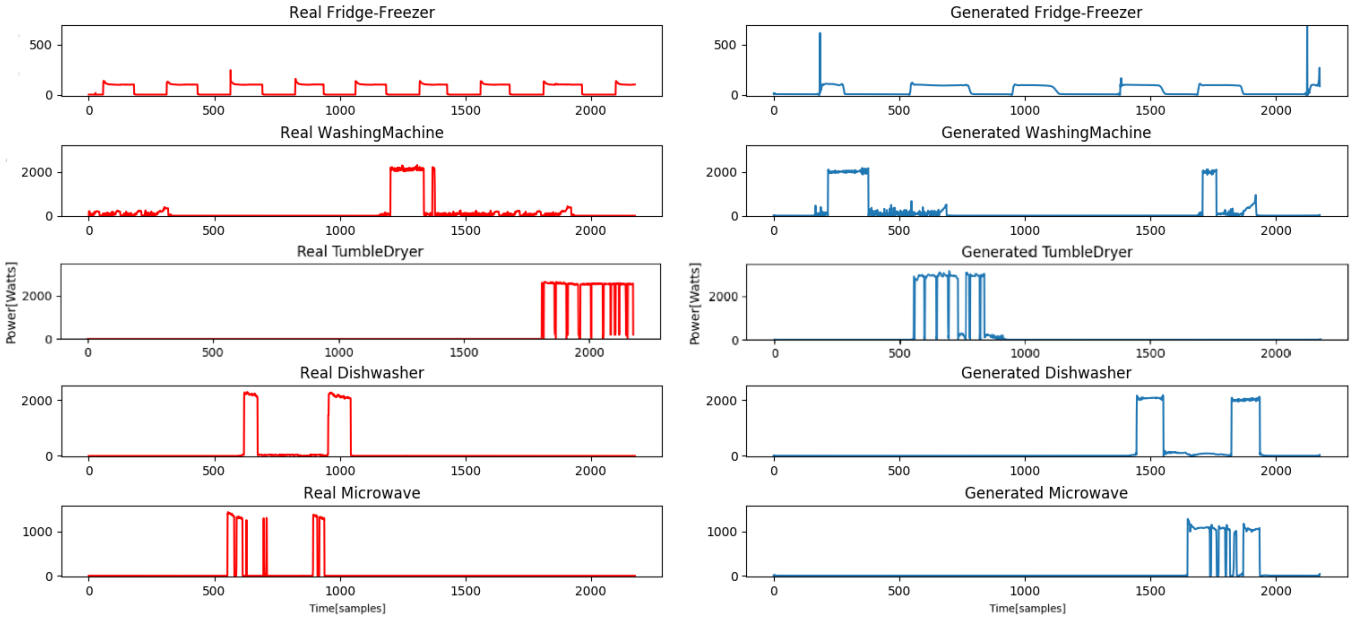

Fig. 4. Examples of appliance power traces generated by TraceGAN, alongside their real counterparts taken from REFIT. We can see here that the generated signals follow the real data closely, yet without direct copying, in important attributes such as power levels, overshoot, quick switching, and more.

standard deviation along the iterations. Table I summarizes the results for all the metrics described above.

For completeness, we have also evaluated each of the appliance classes individually, using the FID, and both variations of the SWD. The IS, by its definition, is not appropriate for the evaluation of instances of a single class and thus is not included in this analysis. Although TraceGAN performs well in the appliance-specific comparisons as well, we believe such appliance-specific metrics fail to paint the full picture. For this reason, we present their result and the corresponding discussion in the Appendix.

TABLE I
SYNTHESIZED APPLIANCE PERFORMANCE EVALUATION

Generator	IS	FID	SWD^*_{Lap}	SWD_{Cl}
Dataset	$3.77 \pm .15$	0	0	0
ANTgen	$3.73 \pm .11$	69.63	$45 \pm .029$	$0.31 \pm .017$
SynD	$3.18 \pm .10$	76.09	$22 \pm .011$	$0.33 \pm .015$
TraceGAN	$\mathbf{3.81 \pm .13}$	**43.30**	$\mathbf{18 \pm .088}$	$\mathbf{0.25 \pm .011}$

*SWD_{Lap} values were calculated using Laplacian "triangle" features were scaled by 10^{-3}. SWD_{Cl} values were calculated using the last layer of classifier features, similarly to the Frechet Inception distance.

Several things stand out when reviewing the quantitative results. First, we notice TraceGAN receives the highest Inception score, outscoring both SynD and ANTgen in a statistically significant manner (t-test $p \leq 1e^{-5}$). TraceGAN even slightly outscores the real data, although not in a statistically significant manner (t-test $p = 0.38$). We believe this is caused by the existence of some inevitably mislabeled data in REFIT. When collecting sub-meter data for NILM applications, the wiring of certain houses makes it difficult to avoid having more than one appliance on each sub-meter. This means that often a sub-meter designated as one appliance (such as fridge or dishwasher) will contain measurements from a smaller, or less commonly used appliance (such as a kettle or battery charger). The presence of such activations may lead to a lower Inception score in the real data, but effects TraceGAN to a lesser extent.

Secondly, we notice that the diversity of TraceGAN-generated signals is noticeable when reviewing the more advanced metrics. In both variations of the SWD as well as FID, TraceGAN outperforms the other two synthesizers in a statistically significant manner (t-test $p \leq 9e^{-4}$). We believe that the combination of these scores shows that TraceGAN is capable of generating samples that are comparable, in terms of realism, with copying or hand-modeling real data directly (as done by SynD and ANTgen), while at the same time creating diverse and truly novel appliance power signatures.

B. Qualitative Analysis

When evaluating our generated signals, we focus on the traces' realism as well as their variety and novelty. We find that TraceGAN is able to generate highly realistic-looking appliance traces while avoiding directly copying existing appliances from REFIT. In addition, we notice that the generator's diversity exists both between classes and within each class.

Fig. 4 shows an example of generated signals from each of the five trained appliances, along with similar real power traces. We can see that the generated signals present highly comparable behaviours and contain all of the major features of each appliance class. Some important attributes in the generated signals are shown below, by class:

- **Fridges** - generated fridge traces maintain the periodic nature of real refrigerators. We see small variation in both frequency and duty cycles of the activations, with minor differences within an activation and larger differences between different samples. In addition, generated fridges maintain the initial spike in power consumption.

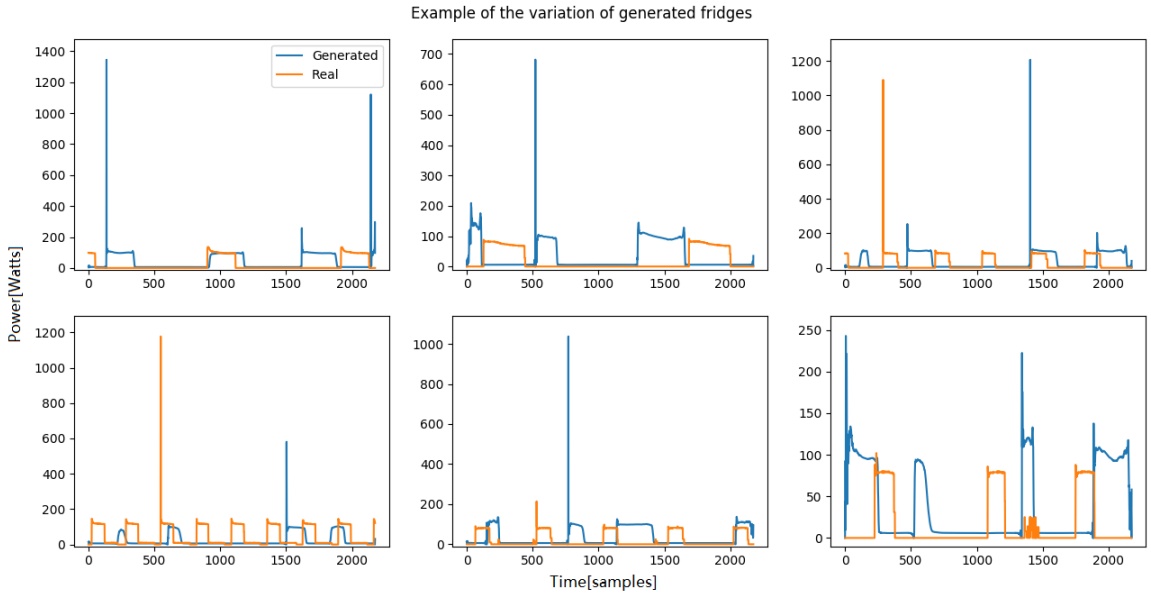

Fig. 5. Examples of generated and real fridges. There is diversity in the generated fridges in terms of frequency, duty cycle, overshoot size, and more. TraceGAN generates some artifacts such as an overshoot at the end of an activation, as well as some power variations within a given activation.

- **Washing Machines** - generated washing machine traces manage to convey the complicated state transitions of the various washing cycle states. We see quick fluctuations in power consumption, typical of the machine's internal heating unit switching on and off. Additionally, the generator is able to generate the variable load which occurs during the washing machine's spin cycle.
- **Tumble Dryers** - generated tumble dryer traces are able to maintain the characteristic drop in power consumption that occurs periodically when the dryer changes direction. Furthermore, TraceGAN is able to capture the usage characteristics of a dryer, occasionally including more than one activation in a 5-hour window.
- **Dishwashers** - generated dishwasher traces manage to maintain the multi-state properties of the original dishwashers, without incurring significant amount of switching noise or any major artifacts.
- **Microwaves** - generated microwave traces portray the low duty cycle of real microwaves, which are generally only used occasionally for periods of a few minutes at most. In addition, TraceGAN is able to generate traces that include quick switching of the microwave oven, which can occur during more advanced microwave modes such as a defrost program.

While TraceGAN generates realistic data for the most part, some issues still exist. The generated signals occasionally contain artifacts that are rare in real signals, such as an overshoot before deactivation, power fluctuations within a given state, or unlikely activation duration. When analyzing these artifacts, we note that examples of such behaviour exist in the real data, albeit rarely. We believe that these behaviours appear in TraceGAN because in the training procedure, such artifacts become central in identifying appliances, leading to them carrying significant gradients to the generator.

In order to demonstrate the diversity of the power traces generated by TraceGAN, we present six examples of generated and real fridge signals in Fig. 5. We note that like the real fridge power traces, the generated signals vary in several important features: power level, activation frequency, duty cycle, and overshoot size. In addition, the generated signals demonstrate some variations in each of the above parameters within an activation window, similarly to real fridges.

C. Data Augmentation Scenario for NILM

One possible use of TraceGAN is as a data augmentation tool for training NILM solutions. In order to demonstrate this, we build a toy scenario using the denoising autoencoder (DAE) method [8], as implemented in the NILMTK-contrib repository [42]. In this scenario, we first create a benchmark by training a DAE for four appliances in the REFIT dataset - refrigerators, washing machines, dishwashers, and tumble dryers. These are 4 of the 5 appliances on which TraceGAN was trained. We then compare these benchmark models with additional models trained using TraceGAN-generated samples for augmentation in several configurations described below.

Since the DAE is trained for each appliance individually, we train four different models for each configuration. Each model is trained on one appliance using data from all houses in REFIT that contained exactly one instance of the corresponding appliance (here we avoid houses with multiple instances for simplicity). The models are trained on the raw aggregate data, without filtering windows into active and non-active ones, and the entire duration of the dataset was used. We train each model for 200 epochs, using the Adam optimizer, with a 90-10 training-validation split of the dataset, saving the model that performs best in terms of validation loss.

Due partly to the properties of the REFIT dataset, the choice of initial learning rate, as well as the relative simplicity of the DAE architecture, the minimum validation loss was achieved quite early for some appliances (e.g., within 15

TABLE II
TRACEGAN AUGMENTED TRAINING RESULTS - REFIT HOUSE 5

Scenario	Fridge			Tumble Dryer			Washing Machine			Dishwasher		
	RMSE	MAE	F1	RMSE	MAE	F1	RMSE	MAE	F1	RMSE	MAE	F1
Benchmark	62.5	43.9	73.5	291.5	97.5	50.1	203.0	48.3	24.0	216.8	43.9	44.6
1% Augmentation	64.8	48.5	70.0	215.9	69.2	60.2	**162.5**	**31.6**	**31.3**	211.8	**43.5**	44.9
2% Augmentation	64.5	46.2	71.4	224.2	71.5	59.2	165.1	34.5	30.8	218.5	44.0	**45.4**
5% Augmentation	**61.6**	**42.9**	**74.0**	**207.6**	**66.5**	**61.3**	165.0	39.4	29.4	**211.4**	47.3	43.8

TABLE III
TRACEGAN AUGMENTED TRAINING RESULTS - RAE HOUSE 1, BLOCK 2

Scenario	Fridge			Tumble Dryer			Washing Machine			Dishwasher		
	RMSE	MAE	F1	RMSE	MAE	F1	RMSE	MAE	F1	RMSE	MAE	F1
Benchmark	67.3	59.2	55.5	**395.0**	132.1	17.2	137.5	50.6	5.9	**370.4**	**116.7**	**5.6**
1% Augmentation	66.8	**57.3**	**55.6**	409.5	**97.6**	**21.4**	**129.6**	**34.0**	**6.7**	372.9	118.1	4.7
2% Augmentation	66.7	57.8	55.5	408.3	120.3	18.3	143.0	42.5	5.9	404.2	125.3	4.8
5% Augmentation	**66.4**	58.0	55.4	426.2	134.2	16.7	189.8	55.4	6.1	387.0	125.1	3.1

epochs for the tumble dryer). Given that the validation set did not include any TraceGAN-generated samples, we would expect that the validation performance of models trained with the augmented datasets would reach their respective maxima earlier due to the relative richness of the augmented training sets. For this reason, we instead terminated training in the augmented scenarios after an additional 10 epochs following the benchmark minimum validation loss.

The augmentation scenarios include randomly replacing p percent of the ground truth appliance data with TraceGAN-generated activations before every training epoch. Note that because most appliances are only used for a small fraction of the time, p must be relatively small (for example, the washing machine is active for approximately 2.5% of the total duration of the REFIT dataset). We then compare the benchmark models and the augmented models for $p \in \{1, 2, 5\}\%$, by evaluating the performance on house 5 in the REFIT dataset (part of which was included in the training set), as well as house 1, block 2 in the RAE dataset [43]. We use the following metrics for comparison: root mean square error (RMSE, in watts); mean absolute error (MAE, in watts), and F1 score (in %). The results are summarised in Tables II and III.

The results in Tables II and III demonstrate the benefits of using TraceGAN data-augmentation in training NILM models. Notably, models trained on augmented datasets outperform the benchmark models in most scenarios, with the dominant augmentation changing slightly between different appliances and datasets. Within the REFIT dataset, we can see the benefits of TraceGAN augmentations as a form of regularization, dissuading the DAE model from overfitting during training. At least to some degree, the benefits of this regularization can be seen when evaluating the models on the RAE dataset, as shown in Table III. It is important to note that attempting to generalize learned models across continents (Europe to North America) is likely an ambitious task relative to the usual use-case for supervised NILM, due to the appliance instances being very different from those seen in training.

Of the two appliances performing with some reasonable degree of accuracy on the RAE dataset (the fridge and tumble dryer), the augmented datasets nearly always provide performance improvements over the benchmark, if only marginally so. On the dishwasher and washing machine, we note that the performance indicates that the instances found in the RAE dataset are vastly different to those found in REFIT. Thus, all results in these scenarios are poor, and any improvement or degradation in performance lacks actual significance.

Important to note is the limitation of the augmentation scenario at the outset: TraceGAN is concerned with appliance-specific generation, not the realistic aggregation of the resulting traces. In this experiment, we added TraceGAN generated windows at random in the dataset, which is highly unlikely to be the optimal augmentation technique. It is likely the case that creating aggregate signals using a combination of TraceGAN and user-behaviour modeling, as used in ANTgen or SynD, will allow additional performance benefits to be gleaned from synthetically augmented datasets.

V. CONCLUSIONS

After identifying the need for synthetic data generation for NILM, we presented here the first GAN-based synthesizer for appliance power traces. Our model, named TraceGAN, is trained in a progressive manner, and uses a unique conditioning methodology to generate multiple appliance classes using one generator. We have also implemented some groundwork for evaluating power trace generators which, as expected, requires more than one metric in order to evaluate the various requirements from synthesizers. Using these metrics, along with visual inspection of the generated samples, we have shown that TraceGAN is able to produce diverse, realistic power appliance signatures, without directly copying or hand-modeling the training data. Additionally, we presented a simplistic scenario for using TraceGAN as a data augmentation tool for training a NILM model.

While the results presented in this paper are based on training on the REFIT dataset, the presented framework can be used for training on any desired dataset, and at any sampling frequency. We believe that these properties may help researchers build upon our simplistic example, and use

TraceGAN as an augmentation tool for training supervised NILM solutions, perhaps in conjunction with the techniques proposed in [20], [21]. The TraceGAN generator can be used to replace certain activation windows in the real data with synthesized ones, with the hope of improving out-of-distribution performance. In order to adapt TraceGAN to fit any augmentation or aggregation methods, one can modify the training procedure of TraceGAN slightly as needed. For example, changing the activation window sizes, or removing the random time shift during training, if a well-localized activation is preferred for disaggregation.

REFERENCES

[1] G. W. Hart, "Nonintrusive appliance load monitoring," *Proc. IEEE*, vol. 80, no. 12, pp. 1870–1891, Dec. 1992.

[2] A. Rodriguez-Silva and S. Makonin, "Universal Non-Intrusive Load Monitoring (UNILM) Using Filter Pipelines, Probabilistic Knapsack, and Labelled Partition Maps," in *2019 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, 2019, pp. 1–6.

[3] Q. Liu, K. M. Kamoto, X. Liu, M. Sun, and N. Linge, "Low-complexity non-intrusive load monitoring using unsupervised learning and generalized appliance models," *IEEE Trans. Consumer Electronics*, vol. 65, no. 1, pp. 28–37, 2019.

[4] S. Makonin, F. Popowich, I. V. Bajić, B. Gill, and L. Bartram, "Exploiting HMM sparsity to perform online real-time nonintrusive load monitoring," *IEEE Trans. Smart Grid*, vol. 7, no. 6, pp. 2575–2585, 2016.

[5] R. Bonfigli, E. Principi, M. Fagiani, M. Severini, S. Squartini, and F. Piazza, "Non-intrusive load monitoring by using active and reactive power in additive factorial hidden markov models," *Applied Energy*, vol. 208, pp. 1590–1607, 2017.

[6] F. M. Wittmann, J. C. Lopez, and M. J. Rider, "Nonintrusive load monitoring algorithm using mixed-integer linear programming," *IEEE Trans. Consumer Electronics*, vol. 64, no. 2, pp. 180–187, May 2018.

[7] M. Z. A. Bhotto, S. Makonin, and I. V. Bajić, "Load disaggregation based on aided linear integer programming," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 64, no. 7, pp. 792–796, July 2017.

[8] J. Kelly and W. Knottenbelt, "Neural nilm: Deep neural networks applied to energy disaggregation," in *Proc. 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, ser. BuildSys '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 55–64. [Online]. Available: https://doi.org/10.1145/2821650.2821672

[9] D. Murray, L. Stankovic, V. Stankovic, S. Lulic, and S. Sladojevic, "Transferability of neural network approaches for low-rate energy disaggregation," in *ICASSP 2019-Int. Conf. Acoust. Spee.* IEEE, 2019, pp. 8330–8334.

[10] J. Kim, T. T. H. Le, and H. Kim, "Nonintrusive Load Monitoring Based on Advanced Deep Learning and Novel Signature," *Computational Intelligence and Neuroscience*, vol. 2017, no. 4216281, 2017.

[11] A. Harell, S. Makonin, and I. V. Bajić, "Wavenilm: A causal neural network for power disaggregation from the complex power signal," in *ICASSP 2019 - Int. Conf. Acoust. Spee.* IEEE, 2019, pp. 8335–8339.

[12] M. Kaselimi, A. Voulodimos, E. Protopapadakis, N. Doulamis, and A. Doulamis, "Energan: A generative adversarial network for energy disaggregation," in *ICASSP 2020 - Int. Conf. Acoust. Spee.* IEEE, 2020, pp. 1578–1582.

[13] Y. Pan, K. Liu, Z. Shen, X. Cai, and Z. Jia, "Sequence-to-subsequence learning with conditional gan for power disaggregation," in *ICASSP 2020 - Int. Conf. Acoust. Spee.*, 2020, pp. 3202–3206.

[14] S. Makonin, B. Ellert, I. V. Bajić, and F. Popowich, "Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014," *Scientific Data*, vol. 3, no. 160037, pp. 1–12, 2016.

[15] D. Murray, L. Stankovic, and V. Stankovic, "An electrical load measurements dataset of united kingdom households from a two-year longitudinal study," *Scientific data*, vol. 4, no. 1, pp. 1–12, 2017.

[16] A. Reinhardt, P. Baumann, D. Burgstahler, M. Hollick, H. Chonov, M. Werner, and R. Steinmetz, "On the accuracy of appliance identification based on distributed load metering data," in *2012 Sustainable Internet and ICT for Sustainability (SustainIT)*. IEEE, 2012, pp. 1–9.

[17] C. Beckel, W. Kleiminger, R. Cicchetti, T. Staake, and S. Santini, "The eco data set and the performance of non-intrusive load monitoring algorithms," in *Proc. 1st ACM conference on embedded systems for energy-efficient buildings*, 2014, pp. 80–89.

[18] N. Buneeva and A. Reinhardt, "Ambal: Realistic load signature generation for load disaggregation performance evaluation," in *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2017, pp. 443–448.

[19] D. Chen, D. Irwin, and P. Shenoy, "Smartsim: A device-accurate smart home simulator for energy analytics," in *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2016, pp. 686–692.

[20] C. Klemenjak, C. Kovatsch, M. Herold, and W. Elmenreich, "A synthetic energy dataset for non-intrusive load monitoring in households," *Scientific Data*, vol. 7, no. 1, pp. 1–17, 2020.

[21] A. Reinhardt and C. Klemenjak, "How does load disaggregation performance depend on data characteristics? insights from a benchmarking study," in *Proc. 11th ACM International Conference on Future Energy Systems (e-Energy)*, 2020.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[23] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[24] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in neural information processing systems*, 2016, pp. 2234–2242.

[25] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[26] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, 2017, pp. 5767–5777.

[27] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[28] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *Proc. 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2642–2651.

[29] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[30] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[31] K. G. Hartmann, R. T. Schirrmeister, and T. Ball, "Eeg-gan: Generative adversarial networks for electroencephalograhic (eeg) brain signals," *arXiv preprint arXiv:1806.01875*, 2018.

[32] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, J. Albrecht *et al.*, "Smart*: An open data set and tools for enabling research in sustainable homes," *SustKDD, August*, vol. 111, no. 112, p. 108, 2012.

[33] S. Henriet, U. Simsekli, G. Richard, and B. Fuentes, "Synthetic dataset generation for non-intrusive load monitoring in commercial buildings," in *Proc. 4th ACM International Conference on Systems for Energy-Efficient Built Environments*, 2017, pp. 1–2.

[34] K. Bao, K. Ibrahimov, M. Wagner, and H. Schmeck, "Enhancing neural non-intrusive load monitoring with generative adversarial networks," *Energy Informatics*, vol. 1, no. 1, pp. 295–302, 2018.

[35] P. O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *Journal of machine learning research*, vol. 5, no. Nov, pp. 1457–1469, 2004.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] S. Makonin and F. Popowich, "Nonintrusive load monitoring (NILM) performance evaluation," *Energy Efficiency*, vol. 8, no. 4, pp. 809–814, 2015.

[38] A. Borji, "Pros and cons of GAN evaluation measures," *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

TABLE IV
SYNTHESIZED SINGLE CLASS APPLIANCE PERFORMANCE EVALUATION

Metric	Generator	Fridge	Tumble Dryer	Dishwasher	Washing Machine	Microwave
FID	TraceGAN	**7.497**	**13.21**	**22.47**	**21.02**	**4.84**
	SynD	37.34	N/A	91.88	50.06	52.6372
	Antgen	33.39	37.45	44.85	133.50	16.9477
SWD_{Lap}	TraceGAN	0.095 ± 0.00027	0.186 ± 0.00065	$\mathbf{0.057 \pm 0.00031}$	$\mathbf{0.030 \pm 0.00033}$	0.102 ± 0.00034
	SynD	$\mathbf{0.0943 \pm 0.000285}$	N/A	0.129 ± 0.00052	0.076 ± 0.00040	0.122 ± 0.00033
	Antgen	0.097 ± 0.00022	$\mathbf{0.104 \pm 0.00056}$	0.077 ± 0.00083	0.102 ± 0.00078	$\mathbf{0.075 \pm 0.00033}$
SWD_{Cl}	TraceGAN	$\mathbf{0.0848 \pm 0.0047}$	$\mathbf{0.135 \pm 0.0085}$	$\mathbf{0.172 \pm 0.010}$	$\mathbf{0.20 \pm 0.015}$	$\mathbf{0.103 \pm 0.0031}$
	SynD	0.238 ± 0.013	N/A	0.422 ± 0.028	0.28 ± 0.014	0.255 ± 0.016
	Antgen	0.221 ± 0.0094	0.223 ± 0.010	0.262 ± 0.017	0.519 ± 0.037	0.151 ± 0.0067

[41] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in neural information processing systems*, 2017, pp. 6626–6637.
[42] N. Batra, R. Kukunuri, A. Pandey, R. Malakar, R. Kumar, O. Krystalakos, M. Zhong, P. Meira, and O. Parson, "Towards reproducible state-of-the-art energy disaggregation," in *Proc. ACM BuildSys*, 2019, pp. 193–202.
[43] S. Makonin, Z. J. Wang, and C. Tumpach, "RAE: the rainforest automation energy dataset for smart grid meter data analysis," *data*, vol. 3, no. 1, p. 8, 2018.

Alon Harell (S'19) received the B.Sc. (*Summa Cum Laude*) degree in electrical and electronic engineering and the B.Sc. (*Summa Cum Laude*) in Physics from Tel Aviv University, Tel Aviv, Israel, in 2012. In 2020, he received the M.A.Sc. degree in electrical engineering from Simon Fraser University, Burnaby, BC, Canada, focusing on deep learning applications for non-intrusive load monitoring. From 2018 to 2020 he was a Research Assistant in the Computational Sustainability Lab. In 2020 he was a MITACS intern with SportLogiq developing graph neural network applications for sports analytics. His research interests include deep learning theory and applications, sustainability, and more recently sports analytics. He was awarded the NSERC CGS-M and PGS-D scholarships, the Graduate Dean Entrance Scholarship, and the British Columbia Graduate Scholarship (Simon Fraser University).

Richard Jones (S'19) received the B.A. degree in psychology and philosophy in 2014, and the B.Sc. (Hons.) degree in physics in 2018, from the University of Manitoba, Winnipeg, MB, Canada, and the M.A.Sc. degree in 2020 from Simon Fraser University, Burnaby, BC, Canada, where he focused on machine learning and electrical engineering. From 2019 to 2020, he was a Research Assistant with the Computational Sustainability Lab, Simon Fraser University. His research interests include the application of machine learning techniques to energy analytics, including non-intrusive load monitoring and demand prediction at various scales. He was the recipient of the NSERC CGS-M, the Graduate Dean Entrance Scholarship (Simon Fraser University), and the Allen Medal in Physics (University of Manitoba).

Stephen Makonin (M'08-SM'13) is an Adjunct Professor in Engineering Science and the Principal Investigator of the Computational Sustainability Lab at Simon Fraser University (SFU). He received his PhD in Computing Science at Simon Fraser University in 2014 in the area of computational sustainability. He has been a software engineer for over 24 years working for various local/international industry clients. Stephen is a registered Professional Engineering (PEng) with Engineers and Geoscientists BC and a Senior Member of the IEEE. His research interests include computational sustainability and the understanding of socioeconomic issues that pertain to technological advancement. Stephen is an expert in data engineering, software engineering, and a world-renowned researcher in non-intrusive load monitoring (NILM) and disaggregation. Stephen is currently the Vice-Chair of the IEEE Signal Processing Society Vancouver Chapter and sits on the IEEE DataPort Advisory Committee. He currently serves as the Editor in Chief of the IEEE DataPort Metadata Review Board, and as an Editorial Board Member of Nature's Scientific Data journal.

Ivan V. Bajić (S'99–M'04–SM'11) received the Ph.D. degree in electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 2003. He is a Professor of Engineering Science and co-director of the Multimedia Lab at Simon Fraser University, Burnaby, BC, Canada. His research interests include signal processing and machine learning with applications to multimedia signal processing, compression, communications, and collaborative intelligence. He is currently the Vice Chair of the IEEE Multimedia Signal Processing Technical Committee and an elected member of the IEEE Multimedia Systems and Applications Technical Committee. He has served on the organizing and/or program committees of the main conferences in the field. He was an Associate Editor of IEEE Transactions on Multimedia and IEEE Signal Processing Magazine, and is currently a Senior Area Editor of IEEE Signal Processing Letters.

APPENDIX
QUANTITATIVE EVALUATION OF INDIVIDUAL APPLIANCES

Table IV shows the results for each metric as evaluated on generated samples from a single class at a time. Note that the SynD was not evaluated for the tumble dryer class as it does not contain any traces of tumble dryers. As we can see, TraceGAN outperforms other generators in 12 of the 15 cases. The only metric in which TraceGAN is sometimes inferior to the other methods is SWD_{Lap}, though it is still performs well in most appliances in that metric. One possible reason for this is that the SWD metric, by its definition, can be biased towards copies of the original data, so long as they somewhat differ from each other (in our case, the windows from each generator are randomly shifted in time). In general, however, it remains clear that TraceGAN is the most consistently reliable and diverse method of all three generators.