ECAI 2024
U. Endriss et al. (Eds.)
© 2024 The Authors.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/FAIA241063

Improving Process Yield Through Manufacturing Digital Twin Using Conditional Synthetic Data Engine (COSYNE)

Shantanu Chandra[a,*], Matthieu Duvinage[b], PKS Prakash[a,], Patrick Davis[a] and Sander Timmer[b]**

[a]ZS, 2nd Floor, MFAR Manyata Tech park, Phase IV, Manayata Tech Park, Nagavara, Bengaluru, Karnataka, India
[b]GSK,20, Avenue Fleming, Wavre, Belgium

Abstract. The pharmaceutical industry must adhere to rigorous regulations to meet specific quality standards. Additionally, the intricate nature of pharmaceutical manufacturing processes and long time to production necessitates timely detection of batch failures. AI/ML models are used for predictive maintenance in an automated and data-driven manner to detect these failures and aid timely intervention. However, these models require substantial amount of data for model training. This can lead to extended time-to-value before a predictive monitoring system can be deployed for any new process due to long process lead times. The current research proposes COSYNE, a generative AI-based approach to generate manufacturing digital twin, reducing the model development time by augmenting synthetic data with real data. The proposed solution is validated on a large pharmaceutical company's batch manufacturing dataset, and the results are benchmarked across multiple dimensions of generation quality. Empirical results demonstrate that the proposed COSYNE outperforms the state-of-the-art approach by 2-3 times on average across all the generation quality metrics. Moreover, COSYNE enhances downstream AI/ML performance significantly through data augmentation and reduces time-to-value by creating high-fidelity digital twins with only 10% of real data and still achieve similar performance as current baseline trained on entire real data.

1 Introduction

The pharmaceutical industry must adhere to rigorous regulations to meet specific quality standard supported by necessary data which must be audit-able by government agencies. Additionally, the intricate nature of pharmaceutical manufacturing processes and long time to production necessitates precise and timely detection of batch failures. Therefore, continuous process monitoring, diagnosis and adjustments are necessary using in-line sensors [32]. In-line with this, the industry is gradually transitioning towards predictive maintenance to pro-actively manage failure and improve yield of these processes [3].

There are three primary stages in drug manufacturing namely drug substance, drug product, and final drug product. The crucial step of drug substance step encompasses cell culture and filtration steps such

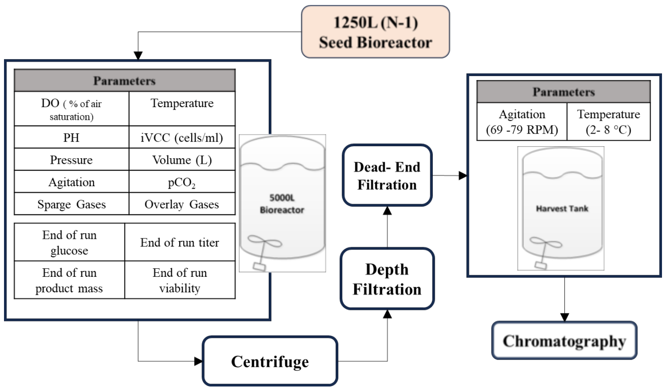

Figure 1. Process flow of the final stage 5000L bio-reactor.

that production in the bioreactor, centrifugation, filtration and harvesting and purification using chromatography is done in sequence. Throughout the process, various parameters are collected via in-line sensors, including temperature, pressure and saturation. At the end of the process multiple key process and end of run parameters such as glucose, titer, product mass or viability are evaluated to understand the quality of the process as illustrated in Figure 1.

Machine learning models play a crucial role in predicting end-of-run parameters early in the production process through in-line sensing. Conventional data-driven predictive maintenance approaches generally require a significant amount of prior knowledge [21]. To address this issue, researchers have explored deep learning based approaches [9] to automate the prediction problem and improve predictability. Leveraging these AI/ML models enables monitoring and early intervention, potentially stopping processes if batch is expected to fail, which can lead to increased process utilization and yield. One of the challenges organizations face while implementing AI/ML models is the requirement for a substantial amount of data for model development. This can lead to extended lead-time before a predictive monitoring system can be built or deployed in production. The current research explores a generative AI-based approach to reduce the model development time by augmenting synthetic data with real data. This research proposes COSYNE (**CO**nditional **SY**nthetic data **E**ngine), an architecture based on Generative Adversarial Networks [16]), aimed at developing a manufacturing digital twin simulator capable of generating synthetic batch data.

* Corresponding Author. Email: shantanu.chandra@zs.com
** Corresponding Author. Email: prakash.prakash@zs.com

Table 1. Related work summary elucidates that while different architectures incorporate few aspects of the desired characteristics, none address *all* of them together. **Stochastic** – model under a probabilistic framework. **Multivariate mixed data types** – model multiple variables of any data type (ordinal, categorical, numeric). **Latent space modeling** – learn in a dense latent space. **Conditional generation (static)** – condition the sequence generation on batch's conditional vectors. **Open loop (training / inference)** – conditioned on its own previous generation at each step. **Closed loop (training / inference)** – conditioned on ground truth sequence at each step.

	Teacher-Forcing [30]	Professor-Forcing [19]	C-RNN-GAN [22]	RC-GAN [13]	TimeGAN [31]	COSYNE (ours)
Stochastic			✓	✓	✓	✓
Multivariate Mixed-Data Types					✓	✓
Latent space modeling					✓	✓
Conditional Generation (static)				✓		✓
Open Loop (training + inference)	✓	✓	✓		✓	✓
Close Loop (training + inference)		✓				✓

The current research focuses on developing manufacturing digital twin using in-line manufacturing sensor and input batch data. The development of a manufacturing digital twin involves several unique challenges related to data and modelling, including: 1) **Personalized generation:** Since sensor output is a function of input batch parameters, generation needs to consider batch input (such as raw materials, previous stage end state, etc) as a conditional parameter during data generation; 2) **Dependency among multivariate time-series:** The process involves handling multivariate time-series data considering auto-correlation and cross-correlation relationships among series; and 3) **Non-homogeneous data resolution:** Due to differences in frequency cycles, data obtained from in-line sensors are not homogeneous and needs to be addressed as part of the design. The current paper focuses on addressing these challenges as part of proposed architecture.

The proposed COSYNE approach has been validated on production process involving 5K bioreactor cell culture batches within a large European pharmaceutical manufacturer's facility. To assess the effectiveness of proposed approach, simulated batches were generated using COSYNE, based on reference historical data. These simulated batches were then enriched with real data, forming the basis of constructing AI/ML models to predict end-of-run parameters. In our evaluation, we compared the performance of simulated data and performance of developed AI/ML model based on COSYNE with the state-of-the-art TimeGAN methodology [31], which specializes in generating multi-variate time-series data. The results of this benchmarking exercise revealed significant advantages of our proposed approach, showcasing improvements ranging from 2 to 3 times across various key metrics as shown in Table 2. These metrics encompassed the fidelity of batch generation, the diversity of outcomes, and the overall utility of the predictive model.

This comparative analysis underscores the robustness and superiority of the COSYNE approach in capturing the complexities of production processes and optimizing predictive modeling within pharmaceutical manufacturing contexts. Rest of the paper is organized as follows: Section 2 presents related prior work; Section 3 describes COSYNE framework; and Section 4 presents the performance of COSYNE using in-line manufacturing from bioreactor process as a case study, followed by conclusions and proposed next steps in Section 5.

2 Related Work

COSYNE is a generative model for discrete-time, multivariate, mixed-type sequence generation. Earliest works in this domain have used variants of Fourier transforms, ARIMA/SMA, dynamic time warping (DTW) or first and second-order Markov chain models [27, 5, 24, 12, 14] for the similar task of time-series generation. However, these methods can not handle multi-variate mixed data types, lack the capability of conditional modeling for personalized generation and do not perform well on panel data (i.e., a collection of data that tracks the behavior of a group of entities over time). Subsequent works adopted auto-regressive (AR) deep learning methods like recurrent neural networks (RNNs). However, AR-RNNs are usually trained via the maximum likelihood (MLE) procedure [29] and are thus prone to predictive error accumulation over long sequences due to discrepancy between closed-loop training (i.e., conditioned on previous step ground truths) and open-loop inference (i.e., conditioned on their own previous step generation). The first set of generative models that gave acceptable performance at the task came from advanced deep learning methods summarized in Table 1. While teacher-forcing [30] and professor-forcing [19] methods addressed the shortcomings of auto-regressive methods, they still did not support sampling from a learned distribution. Thus, the only source of variability in output could be derived only from the output probability model. Multiple subsequent studies such as RC-GAN [13] and C-RNN-GAN [22] inherited the GAN [15] framework to introduce probabilistic learning paradigm for temporal data using adversarial learning. Both these methods rely solely on unsupervised adversarial learning which the current SOTA method, TimeGAN [31], empirically demonstrated to not be sufficient to model the underlying data. TimeGAN outperformed all the above approaches by adopting supervised as well as unsupervised training of the network components. However, even TimeGAN does not support conditional generation capability in its experiments, and also suffers adversely with mode-collapse (the phenomenon of generating similar sequences repeatedly and failing to generate diverse realistic samples).

Our proposed method tackles these existing shortcomings by modeling the batch meta-data (raw materials, initial conditions, etc) as conditional inputs to generate realistic manufacturing process progression. Additionally, to enable personalized generation and stabilize GAN training, it introduces a diversity-enhancing adversarial objective. Finally, unlike previous works, COSYNE supports multiple generation scenarios encountered in real-world use cases in a single framework (i.e., pure generative mode as well as forecasting from partial sequences; discussed in detail in Section 3.2.2).

3 Methodology

3.1 Problem Formulation

Let $\mathcal{X}_s$ and $\mathcal{X}_t$ represent the vector space of the static conditional features (i.e., batch meta-data) and temporal features of batches (temperature, pressure, pH, etc) respectively. Let $\mathbf{x}_s$ and $\mathbf{x}_t$ be instances of specific values of these vector spaces. Note that $\mathbf{x}_s$ and $\mathbf{x}_t$ are vectors consisting of variables of mixed data types (continuous, categorical and ordinal).

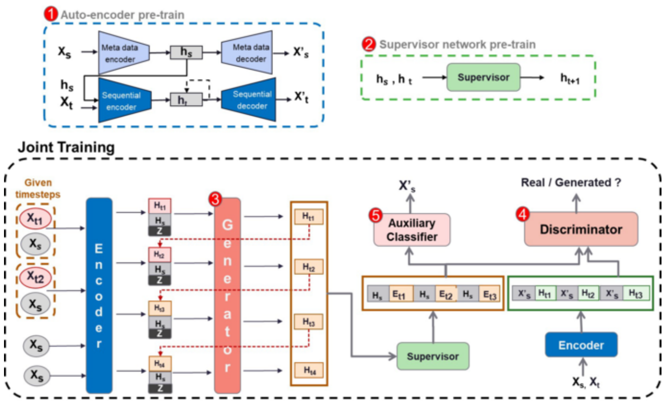

Figure 2. COSYNE architecture diagram detailing the five network components and training stages.

To extend this over a collection of batches, let the individual samples of data be indexed by $n \in 1, ..., N$, so we can denote the batch manufacturing dataset as $\mathcal{D} = (\mathbf{X}_{n,s}, \mathbf{X}_{n,1:T_n})$. Going forward, subscripts n are absorbed in the notation and omitted unless explicitly required. Our goal is to use training data $\mathcal{D}$ to learn a joint density $\hat{p}(\hat{\mathbf{X}}_s, \hat{\mathbf{X}}_{1:T})$ that best approximates the original joint density $p(\mathbf{X}_s, \mathbf{X}_{1:T})$ that represents how do the static and temporal variables interact with each other to dictate the progression of batch processes. To do so, we make use of the autoregressive decomposition as $p(\mathbf{X}_s, \mathbf{X}_{1:T}) = p(\mathbf{X}_s) \prod_t p(\mathbf{X}_t | \mathbf{X}_s, \mathbf{X}_{1:t-1})$, i.e., target joint distribution of batch processes can be jointly learned using the input batch meta-data ($\mathbf{X}_s$) and the batch progression so far (summarized by $\mathbf{X}_{t-1}$) to determine state in the next step ($\mathbf{X}_t$) iteratively. For the task of conditional generation, the static data ($\mathbf{X}_s$) is always assumed to be given, meaning, $p(\mathbf{X}_s)$ is known. This reduces the learning objective to approximating the real $p(\mathbf{X}_t | \mathbf{X}_s, \mathbf{X}_{1:t-1})$ by the learned $\hat{p}(\hat{\mathbf{X}}_t | \hat{\mathbf{X}}_s, \hat{\mathbf{X}}_{1:t-1})$ for all timesteps t. Under a GAN framework, this approximation objective takes the form of the Jensen-Shannon Divergence (JSD) between the real density (p) and learned density ($\hat{p}$) yielding the objective: $min_{\hat{p}} JSD\big(p(\mathbf{X}_t | \mathbf{X}_s, \mathbf{X}_{1:t-1}) || \hat{p}(\hat{\mathbf{X}}_t | \hat{\mathbf{X}}_s, \hat{\mathbf{X}}_{1:t-1})\big)$. Moreover, we transfer the entire learning procedure into a lower-dimensional manifold to aid the model to aid learning. We achieve this via an auto-encoder which helps with feature compression while preserving the underlying relationship between the variables. This switch in formulation to latent space is achieved by a minor change in the aforementioned learning objective, such that $\mathbf{X}_s$, $\mathbf{X}_{1:T}$ are replaced with their hidden representations $\mathbf{h}_s \in \mathcal{H}_s$, $\mathbf{h}_{1:T} \in \mathcal{H}_t$ respectively, where $\mathcal{H}_s$ and $\mathcal{H}_t$ represent the corresponding latent vector spaces.

3.2 Proposed Architecture: COSYNE

COSYNE has two pivotal design aspects, a) the mapping of the learning process into a lower-dimensional manifold. This enables all components, both supervised (*AE,S*) and adversarial (*G,D,AuxC*), to operate on latent representations; and b) introducing a novel auxiliary classifier unit to aid conditional learning. The 1) *auto-encoder (AE)* maps data from the feature space to the latent space ; 2) the *supervisor network (S)* aids the generator through a supervised objective, enhancing the learning process ; 3) *conditional generator (G)* auto-regressively produces batch simulations ; 4) the *conditional discriminator (D)* enables the unsupervised learning objective of the generator by providing adversarial feedback; 5) the *auxiliary classifier (AuxC)* supports the generator in understanding the relationship between input conditions and their sequences.

3.2.1 Auto-Encoder (AE)

COSYNE uses a sequential auto-encoder(AE) [7, 26] to map the input data from the original data space to a lower dimensional manifold and back. MedGAN and MedWGAN [8, 2] used auto-encoders for high dimensional cross-sectional medical data compression. The proposed COSYNE extends this concept to spatio-temporal data. Specifically, the encoder transforms static and temporal features to their latent representation, i.e., $Enc : \mathcal{X}_s \times \prod_t \mathcal{X}_t \rightarrow \mathcal{H}_s \times \prod_t \mathcal{H}_t$ such that: $h_s = Enc_s(\mathbf{s})$, $h_t = Enc_t(\mathbf{h}_s, \mathbf{h_{t-1}}, \mathbf{x_t})$,

where, $Enc_s : \mathcal{X}_s \rightarrow \mathcal{H}_s$ is the encoder for static features, $Enc_t : \mathcal{H}_s \times \mathcal{H}_t \times \mathcal{X}_t \rightarrow \mathcal{H}_t$ is the encoder for temporal batch processes, and $\mathbf{s}, \mathbf{x_t}$ are the static and temporal data in original space respectively. Note that static features are encoded independently, however, temporal variables are encoded by auto-regressively conditioning on previous time-step information as well as the corresponding static conditions of the said temporal sequence. Similarly, the decoder reverse maps the learned latent representations to original data space, i.e., $Dec : \mathcal{H}_s \times \prod_t \mathcal{H}_t \rightarrow \mathcal{X}_s \times \prod_t \mathcal{X}_t$ such that $\tilde{x}_s = Dec_s(\mathbf{h}_s)$, $\tilde{x}_t = Dec_t(\mathbf{h}_t)$, where, $Dec_s : \mathcal{H}_s \rightarrow \mathcal{X}_s$ is the decoder for static features, and $Dec_t : \mathcal{H}_t \rightarrow \mathcal{X}_t$ is the decoder for temporal batch processes. We use multi-layer perceptrons (MLPs) as static components of the AE, while GRU [6] (a form of RNN) as the temporal AE units. Note that these encoders and decoders can be parameterized by any architecture of choice that are auto-regressive in nature that do not violate the causal ordering of information. This causal ordering constraint on the architecture ensures that output at each time-step depends only on the previous time-step information to prevent information leakage.

3.2.2 Conditional Generator and Discriminator (G and D)

Since we transfer the entire learning process into a lower dimensional manifold, the generator(G) does not produce synthetic samples directly but rather their latent representations. Subsequently, the discriminator (D) is trained to distinguish the latent representations of real data from the generated latent vectors. GANs learn by mapping a known distribution to the learned target distribution via sampling random vectors. Let $\mathbf{Z_t} \in \mathcal{Z}_t$ be the sampled random vector from a known distribution space, then $G : \prod_t \mathcal{Z}_t \rightarrow \prod_t \mathcal{H}_t$ such that: $\hat{\mathbf{h}}_t = G(\mathbf{h}_s, \hat{\mathbf{h}}_{\mathbf{t-1}}, \mathbf{z_t})$ where $\mathbf{h}_s$ is the latent representation of input static vector obtained from AE component E_s used for conditional generation and $G : \mathcal{H}_s \times \mathcal{H}_t \prod_t \mathcal{Z}_t \rightarrow \mathcal{H}_t$ is the auto-regressive generator of COSYNE which we parameterize with a GRU network. We sample the noise vector z_t from a standard Gaussian distribution using the Wiener process [20]. Subsequently, the discriminator receives the static and temporal representations $D : \mathcal{H}_s \times \prod_t \mathcal{H}_t \rightarrow \mathbb{R} \in [0, 1] \times \prod_t \mathbb{R}_t \in [0, 1]$: $\tilde{y}_t = D(\tilde{\mathbf{h}}_s, \tilde{\mathbf{h}}^*_t)$

where $\tilde{\mathbf{h}}^*_t$ notation indicates either real ($\mathbf{h}_t$) or generated ($\hat{\mathbf{h}}_t$) temporal latent vectors.

While the G network follows the auto-regressive principles, the D network isn't bound by such causal ordering constraints. G can only generate future steps based on past context, whereas D evaluates the entire sequence at once, leveraging both past and future context to provide detailed feedback to G. To that end, we employ a bi-directional GRU with an MLP classification layer for D, allowing it to assess sequences from both directions for improved classification. Furthermore, due to its conditional auto-regressive design, COSYNE offers two types of generation: *Type A* generates entire batch process end-to-end from conditional vectors, useful for replicating datasets with privacy safeguards; *Type B* simulates or imputes missing journey segments with partial time-step information, ideal for trial monitoring with increasing data precision. This flexibility allows COSYNE to adapt to various real-world scenarios based on available inputs.

3.2.3 Auxiliary Classifier (AuxC)

COSYNE aims to condition simulations on batch-level conditional features (raw materials, batch meta-data, etc). The discriminator distinguishes between real and generated sequences but doesn't explicitly guide coherent sequence generation with static conditionals, causing mode collapse (where the generator produces similar sequences regardless of input conditionals due to a lack of learned relationships between input conditions and output sequences). To tackle this issue, we employ the auxiliary classifier (AuxC) to assist the generator in learning the joint distribution $\hat{p}(\hat{\mathbf{X}}_t|\hat{\mathbf{X}}_s, \hat{\mathbf{X}}_{1:t-1})$ much more effectively. Specifically, AuxC has the opposite objective than the generator, as it predicts the input conditional vector given a sequence of latent journey representations such that, $AuxC: \prod_t \mathcal{H}_t \rightarrow \mathcal{H}_s$: $\mathbf{H}_s = AuxC(\tilde{\mathbf{h}}^*_T)$ where $(\tilde{\mathbf{h}}^*_T)$ denotes either real $(\mathbf{h}_T)$ or generated $(\hat{\mathbf{h}}_T)$ final step latent representation of batch processes. We parameterize AuxC with a MLP layer. This work adapts this concept from image synthesis literature [23, 18] to the spatio-temporal setting, by introducing a novel auxiliary classifier setup for sequential auto-regressive architectures which has not be shown to work before. Unlike in image synthesis where the input is a specific class, here the input condition is a complex multivariate vector mapped into the latent space. The introduced AuxC in COSYNE explicitly connects parts of the latent space to model inputs, establishing a direct conditional relation between the input and output.

3.2.4 Supervisor Network (S)

Pure unsupervised adversarial loss (from D and AuxC) may not be enough for the generator to learn the spatio-temporal relations efficiently. Thus, we introduce the supervisor network (S) to further discipline the training and assist the generator with a direct supervised loss for explicit feedback in its auto-regressive generation process. Specifically, $S: \mathcal{H}_s \times \prod_t \mathcal{H}_t \rightarrow \mathcal{H}_t$, such that: $\mathbf{h}_t = S(\mathbf{h}_s, \mathbf{h_{t-1}})$ Thus, the role of the supervisor network is to input information at time step $t-1$ to produce data at t. This is leveraged during the generator training phase where generated sequences are fed to the supervisor to compute supervised loss on the generator's outputs. We parameterize S with a GRU network.

3.3 Multi-objective Training

COSYNE is trained using supervised, semi-supervised as well as unsupervised learning paradigms. As shown in Figure 2, we split the training pipeline into two steps. The first step is to pre-train the individual components of COSYNE on their respective objectives to infuse them with a preliminary knowledge of their tasks. The second step is to jointly train all the networks together to align their latent spaces into a common manifold to achieve the overall conditional generation objective.

The auto-encoder is trained on reconstruction loss, which is realized by mean squared error (MSE), such that: $\mathcal{L}_R = \mathbb{E}_{x_s, x_{1:T}\sim p}\Big[||x_s - \tilde{x}_s||_2 + \sum_t ||x_t - \tilde{x}_t||_2\Big]$.

The original GAN training objective is designed to predict how real an input is $(D(x))$. However, this objective suffers from vanishing gradients problem early on in the training. Thus, we employ the relativistic GAN (RSGAN) objective [17] to tackle this issue and stabilize GAN training. The RSGAN computes a "distance" $D(x_r, x_f)$, i.e., the probability that the real data is more realistic than the fake data which gives us the unsupervised adversarial objective to be $\mathcal{L}_U = L_D^{RSGAN} + L_G^{RSGAN}$:

$$\mathcal{L}_U = -\mathbb{E}_{(h_s, h_{1:T}\sim p)\,;\,(\tilde{h}_s, \tilde{h}_{1:T}\sim\tilde{p})}\Big[\sum_t \log(\sigma(y_t - \tilde{y}_t)) + \sum_t \log(\sigma(\tilde{y}_t - y_t))\Big] \quad (1)$$

As mentioned in Section 3.2.2, COSYNE is capable of *Type A* and *Type B* forms of generation scenarios based on the real-world use-cases. We incorporate this capability in the training dynamics as well, by training the auto-regressive conditional generator in closed-loop (generative) mode for *Type A* and open-loop (teacher-forcing) mode for *Type B*. In the open loop mode, the generator is recursively fed with temporal information from its own generation of the previous step, i.e., $p(\mathbf{H}_t|\mathbf{H}_s, \mathbf{H}_{1:t-1})$ is approximated by $\hat{p}(\hat{\mathbf{H}}_t|\mathbf{H}_s, \hat{\mathbf{H}}_{1:t-1})$. Meanwhile, in the open loop mode, the generator is fed with partial sequences of real data journey at each step, i.e., it is approximated by $\hat{p}(\hat{\mathbf{H}}_t|\mathbf{H}_s, \mathbf{H}_{1:t-1})$. The unsupervised loss function remains the same post this subtle change in the generation process.

The AuxC network is trained to predict the original conditional latent vector (coming from real data) given the batch process progression. This is achieved by training the AuxC on reconstruction loss realized via MSE such that: $\mathcal{L}_{AuxC} = \mathbb{E}_{(h_T\sim p)\,;\,(\hat{h}_T\sim\tilde{p})}\Big[||h_T - \hat{h}_T||_2\Big]$

Finally, to assist the generator with an additional supervised loss, the supervisor network takes the inputs of the generator to predict the real next-time-step information. The gradients are thus computed on MSE supervised loss that measures the discrepancy between the distributions $p(\mathbf{H}_t|\mathbf{H}_s, \mathbf{H}_{1:t-1})$ and $\hat{p}(\hat{\mathbf{H}}_t|\mathbf{H}_s, \hat{\mathbf{H}}_{1:t-1})$ such that: $\mathcal{L}_S = \mathbb{E}_{x_s, x_{1:T}\sim p}\Big[\sum_t ||h_t - G(h_s, h_{t-1}, z_t)||_2\Big]$ where $G(h_s, h_{t-1}, z_t)$ approximates $\mathbb{E}_{z_t\sim\mathcal{N}}[\hat{p}(\hat{\mathbf{H}}_t|\mathbf{H}_s, \hat{\mathbf{H}}_{1:t-1})]$. In essence, during the training phase, COSYNE's task is achieved by disciplining the generator via the unsupervised $(\mathcal{L}_U)$ as well as supervised losses $(\mathcal{L}_S)$. While $\mathcal{L}_U$ assists the generator in learning to produce realistic spatio-temporal sequences (assessed by an imperfect adversary in a minmax game), the $\mathcal{L}_S$ pushes the generator to be consistent in its step-wise auto-regressive generation.

Optimization: The first phase of COSYNE training involves pre-training all the supervised components (AE, S) on their respective objectives as a warm-up. In the second phase, both the supervised and unsupervised components are trained jointly under unified objectives as detailed below.

Let $\theta_{enc}, \theta_{dec}, \theta_S, \theta_G, \theta_D, \theta_{AuxC}$ denote the parameters of the components respectively. Then AE is trained as:

$$\min_{\theta_{enc}, \theta_{dec}} (\lambda\mathcal{L}_S + \mathcal{L}_R) \quad ; \quad \min_{\theta_T} \mathcal{L}_T \quad (2)$$

,where, $\lambda \geq 0$ is the relative weight of supervised loss in the objective. The joint training is based on the updated input space coming from AE and G components, since they are brought into a common latent manifold. Finally, the conditional generator and discriminator with auxiliary classifier are trained adversarially as:

$$\min_{\theta_G}\Big(\eta\mathcal{L}_S + \max_{\theta_D, \theta_{AuxC}} (\mathcal{L}_U + \delta\mathcal{L}_{AuxC})\Big) \quad (3)$$

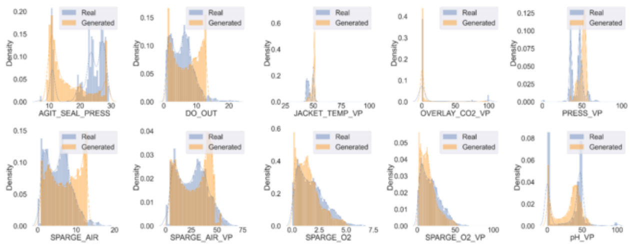

Figure 3. KDEs of COSYNE show good capture of real data variance.

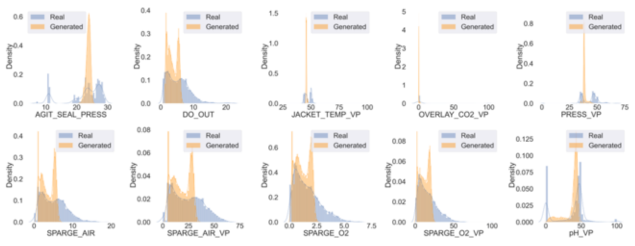

Figure 4. KDEs of TimeGAN show it struggles to capture the data variance well.

where, $\eta, \delta \geq 0$ are the relative weights of supervised and auxiliary loss respectively. This form of multi-objective training equips COSYNE to encode feature vectors into a joint latent space and generate latent representations while conditioning the entire process on batch meta-data.

During our experiments we note that COSYNE is not sensitive to λ, η and δ, as the training dynamics do not vary substantially based on the relative re-weighting of the respective losses. Furthermore, using the RSGAN objective helps to tackle the vanishing gradients problem, auxiliary classifier along with supervised training help in preventing mode collapse.

4 Results and Analysis

4.1 Data

We perform our experiments using a large European pharmaceutical manufacturer's Upper Merion 5K Nucala Hackathon dataset. The dataset comprises of manufacturing of a drug produced in a stirred-tank cell culture process. The dataset contains sensor readings such as pH, pCO2, dextrose, lactate, VCC, % viability, and titer at minute-level throughout the processes (300L, 1250L and 5000L bio-reactors) of 180 batches. For the scope of our experiments we focus on the 5000L bio-reactor which is the last stage in manufacturing and the previous steps are mainly used for cell culture development. We have 10 time-varying variables of 5000L in scope from sensor readings and 12 conditional variables as batch meta-data in our experiments as detailed in Table 1 of supplementary material [4]. The batch meta-data variables represent the end state of the previous stage of each batch, along with the raw material quantities used at the start of the current stage (5000L bio reactor). The time-series variables were aggregated at a hourly level to suppress noise, post which the average sequence length of batch time series data was 348.8 time steps.

4.2 Benchmarks and Evaluation

Benchmarks: The authors of TimeGAN have extensively benchmarked it with the works such as RC-GAN [13], C-RNN-GAN [22],

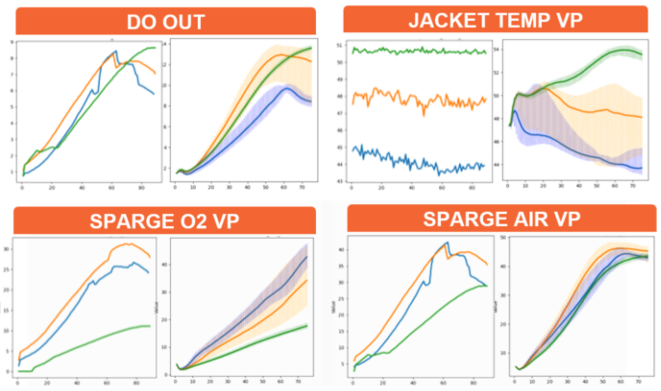

Figure 5. Simulation results of 3 random batches denoted by 3 colors. Real batch data (left) and digital twin simulations from COSYNE for the same batch (right) show similar progression patterns across the variables.

and shown it to outperform all of them to be the current SOTA. Additionally, TimeGAN comes the closest to our proposed framework in terms of required capabilities for the task of batch processes simulation (Table 1). Thus, we benchmark the performance of COSYNE against the current SOTA TimeGAN in this work. To prevent information leakage and mimic real-world scenarios of working on unseen data, we make train and test splits (80% train and 20% test) of the data and train the generative models *only* on the train set. The generative models are then tested on the unseen test set over 1000 simulations for each batch in the test set (on metrics mentioned below), to assess their generalizing capability to unseen data. All the subsequent analysis from hereon for all the models is based on the held-out test set. We report all the performance scores in Table 2.
Evaluation: We holistically evaluate the aforementioned models on three dimensions of evaluations that represent desirable generation quality - a) *fidelity*, b) *diversity* and, c) *utility*.

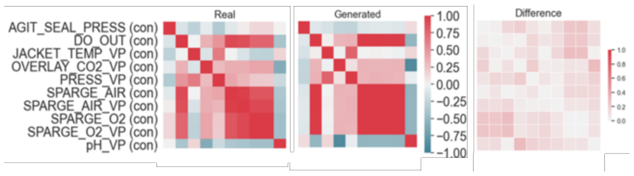

Figure 6. COSYNE captures the correlation very accurately

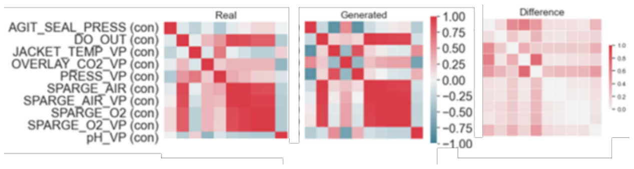

Figure 7. TimeGAN struggles to model the underlying correlations.

Generation Fidelity: The fidelity of generated data entails how coherent are the generated processes in terms of inter-variate correlations and adherence to batch meta-data resulting in generated samples being indistinguishable from real ones. We use 5 fidelity metrics:

1. Univariate KDEs: to test if the model can capture the multiple modes of each of the mixed-type variables well, we use kernel density estimate (KDE) plots. Figure 4 and Figure 3 show the KDE plots for TimeGAN and COSYNE respectively. We observe that the mean as well as the variance across all the variables is much more accurately modeled by COSYNE than TimeGAN. COSYNE shows

Table 2. Generation quality metrics results over 1000 simulations for each batch in the test set. Values in **Bold** represent best scores. ↑ denote higher scores are better for that metric, and ↓ the vice versa.

Model name	ACD ↓	FID ↓	Alpha-PR ↑	Discriminator Score ↓			Next-step Prediction ↓	
				AUCROC	PRAUC	F1	Real MSE	Gen MSE
TimeGAN [31]	0.18	2.27	0.20	0.70	0.68	0.70	0.31	0.11
COSYNE	**0.11**	**1.15**	**0.78**	**0.55**	**0.57**	**0.58**	0.31	**0.29**

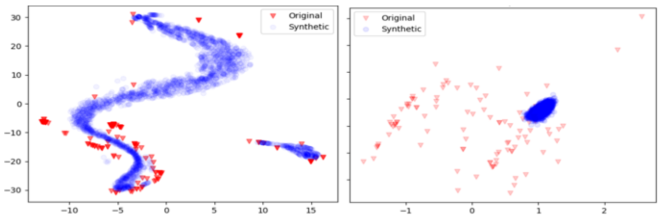

Figure 8. tSNE plots of COSYNE(left) and TimeGAN(right) show that COSYNE is capable of more accurate conditional generation of batch sequences, and hence offers more diversity

to consistently capture the long tail of the distributions quite well, which is typically hard to model in low resource settings and becomes a failure case for most AI/ML models.

2. Batch simulations: to see how different batches progress in real life and how closely do their corresponding digital twin simulations from COSYNE look like we sample 3 random batches and present their actual vs simulated progressions for the time-varying features. Figure 5 shows that

3. Correlation heatmaps: next, we assess how well the model captures the bivariate correlations between these variables of underlying real data. Since we are dealing with mixed-type data, we employ multiple correlation strategies for different type pairs: a) *Continuous-continuous*: Pearson's correlation [11], b) *Continuous-categorical*: Correlation ratio [11] and c) *Categorical-categorical*: Theil's U correlation [25]. To quantify the quality of correlations captured, we define $Avg\_corr\_diff = \frac{1}{p}\sum_{i,j}|corr_R[f_i, f_j] - corr_G[f_i, f_j]|$, where p denotes the total no. of bi-variate feature combinations, i, j denote the pair of features, and $corr_*[f_i, f_j]$ denotes the correlation function (one of the three mentioned above depending on i, j). Lower value of Avg_corr_diff indicates better generation. In Figures 6 and 7 we visualize the correlation heatmaps for the a) original data, b) generated data of TimeGAN and COSYNE, as well as c) the difference between the real and generated correlations for each model. Lighter shades of difference plots indicate better correlation captured by the model. We clearly note that COSYNE captures the underlying correlations much more closely than TimeGAN. This is also reflected in the quantified Avg_corr_diff metric as reported in Table 2.

4. t-Distributed Stochastic Neighbor Embedding (tSNE) [28]: is used to visualize high-dimensional data that lie on several different, but related, low-dimensional manifolds. We visualize the real and generated embeddings in a common 2D space to assess how close do the real and generated processes lie in the latent space. In Figure 8, each red point represents the 2-D projection of a batch's process, while blue points represent generated sequences. We see that COSYNE captures multiple distinct regions of data well, exhibiting better diversity. On the other hand, TimeGAN exhibits mode collapse and results in generating similar processes around a small region of latent space. This demonstrates that COSYNE generalizes much better to unseen data due to its conditional generation capability.

5. Frechet Inception Distance (FID) [10]: is a measure of similarity between two distributions while also taking their multi-variate correlation into account such that: $FID_{RG} = ||\mu_R - \mu_G||^2 - Tr(\Sigma_R + \Sigma_G - 2\sqrt{\Sigma_R\Sigma_G})$ where R and G are the real and generated data embeddings, μ_R and μ_G are the magnitudes of R and G, Tr is the trace of matrix and Σ_R and Σ_G are the covariance matrices of the vectors. FID is bounded between $[0, \inf]$ and since it is a distance measure, a lower FID score implies smaller distances between real and generated samples, thereby indicating better generation quality. As seen from Table 2, COSYNE significantly outperforms TimeGAN on FID scores.

5. Discriminator score (D-score): is the sequence classifier's performance that distinguishes real from generated samples, assessed using F1, PrAUC, and AUCROC metrics. Scores around 0.5 indicate difficulty in differentiation between real and generated, suggesting superior generation. Table 2 shows COSYNE's ability to generate batch processes indistinguishably, with scores approaching 0.5, affirming its superior conditional generation capability.

4.2.1 *Generation Diversity*

Deep generative models, especially GANs, are susceptible to the phenomenon of mode collapse (generate the same process repeatedly, but generate it really well). Thus, the generation diversity metric evaluates how well can multiple types of processes present in the data being generated.

6. Alpha-Precision (Alpha-Pr) [1]: is the fraction of synthetic samples that resemble the most typical α fraction of real samples. It measures the likelihood of a private sample y_g belonging to the real distribution $\mathcal{D}_{Real}$ within its α-support (e.g. a random sub-set of the real data, determined by α-mass). Higher scores imply better distribution overlap between real and generated data indicating the model captures all the modes of $\mathcal{D}_{Real}$. We report alpha-Pr scores for $\alpha = 0.95$ in Table 2. We observe that COSYNE covers 78% of the real data, while TimeGAN manages to cover just 20% of the real data diversity. The TimeGAN architecture thus suffers severely from mode collapse, while COSYNE successfully navigates this issue owing to its architecture design.

4.2.2 *Generation Utility*

The generated data should match the performance of real data at downstream ML tasks to demonstrate its fidelity in real-world use-cases.

7. Next-step Prediction (NxsP): the utility of the generated batches is evaluated by a downstream ML-task of next time-step prediction where historical data is used $[\mathbf{X}_s, \mathbf{X}_{1:t-1}]$ to predict time dependent variables at the next time-step $\mathbf{X}_t$. We use an independent post-hoc GRU network for this task. We thereby train NxsP models $NxsP_R$ and $NxsP_G$ on real train set D_{Rtrain} and generated data D_{Gtrain}, and finally test the performance of both the models on the common real test set, D_{test}. From Table 2 we can see that COSYNE again significantly outperforms TimeGAN by having performance closest to the models trained on real data. This demonstrates that generated data from COSYNE very closely replicates the properties of real data, thus data generated via COSYNE can be used as a close substitute to the real data for any analytics use case.

4.3 Data Augmentation for Yield Prediction

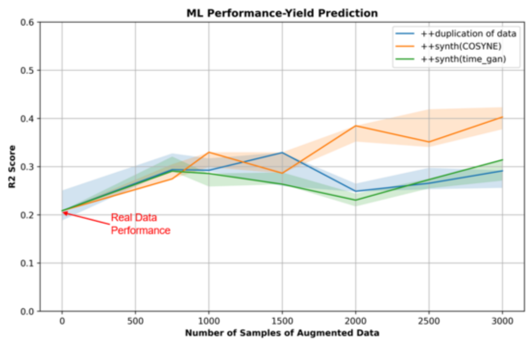

Figure 9. ML performance results for data augmentation using the duplication baseline, TimeGAN and COSYNE. The shaded regions denote the standard deviation over multiple runs.

The real batch data is usually scarce and obtained over long lead times which is not enough to be consumed by ML models for optimal results. However, it can be augmented with synthetic samples to improve downstream analytics. To test the efficacy of data augmentation, we first perform the same ML task with real data alone, followed by combining real data with additional data coming from 3 augmentation methods a) COSYNE, b) TimeGAN and b) a simple data duplication baseline. The ML task for evaluation is predicting end of run titer, for which we leverage a post-hoc independent XGBoost model. We evaluate R2 (higher the better) at different degrees of augmentation, i.e., adding progressively more synthetic samples from each method to the fixed real data samples. We Note that generating synthetic samples from a trained model is a simple inferencing step and is not compute extensive (i.e, generating 100 samples vs 10000 samples takes almost the same time and compute resources). From Figure 9 we observe that by using 0 augmented samples, i.e, just real data, the R2 achieved is 0.2 on the test set. This improves to up to 0.32 on simply adding 1500 duplicated real data points post which adding more duplicates hurts the ML performance due to overfitting. TimeGAN also achieves the best R2 of 0.29 at 750 augmentation samples, post which adding more samples sees no change in performance. However, adding more synthetic samples from COSYNE progressively makes the downstream task better achieving the best R2 of 0.4 at 3000 samples post which we observed no improvements. This shows that augmenting real data with synthetic simulations from COSYNE doubles the performance on unseen test set compared to real data alone. This is because the simulated samples represent valid batch progressions and relay unseen information to the ML model in addition to what real data provides. This can also be seen from the tSNE plots in Figure 8, where the synthetic batches (blue points) are interpolated between various regions of real data (red points). These interpolated regions represent the possibilities of real batches lying here, had more real data been available. This aids the ML models trained on COSYNE augmented data to generalize well to unseen test data.

4.4 Data Augmentation with limited training data

A critical drawback of real batch manufacturing data is that it takes a long time to be made available due to long lead times of the processes that can span even months. This delays time-to-value as many downstream analytics tasks can not run unless sufficient population of batches are available for modeling. With the promise seen in Section

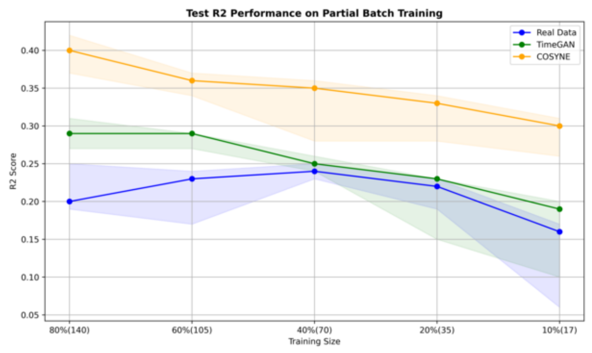

Figure 10. ML performance results for the three models when training with progressively less real data.

4.3 in improving ML performance using data augmentation, we test if we can generate high fidelity synthetic data even with few available real data batches to train on. This will help to expedite the ML tasks thereby improving the future batches' yield by taking corrective steps based on digital twin simulations. To test the fidelity of digital twins created from few data points, we train COSYNE and TimeGAN on progressively less training data (70%, 60%, ..., 10%), and duplication baseline also replicates fewer unique batch data in each of the iteration. We then use the synthetic samples from these methods (1500 for duplication, 1000 for TimeGAN, 3000 for COSYNE) to augment the real data and report R2 across different scenarios. From Figure 10 we see that overall downstream ML performance drops for all the models as we use less real data to train the synthetic data generators on. However, we see that while TimeGAN performance drops by 38% to 0.19 R2, COSYNE drops by just 25% to 0.30 R2 when we train on just 10% data instead of 80%. It is also worthy to note that COSYNE achieves similar performance with just 10% data that TimeGAN does with 80% real data. This attests to COSYNE's ability to learn effectively even in low resource scenarios and simulate high fidelity digital twins leveraging its conditional generation ability. This capability can a) drastically reduce time-to-value in all analytics use cases and b) enable more batches with optimized yield to be produced thereby increasing the efficiency of the production line significantly.

5 Conclusion

In this work, we proposed COSYNE, a deep generative method capable of generating realistic batch manufacturing digital twins that exhibit sensor progression characteristics consistent with real batches. We empirically show that COSYNE outperforms the current SOTA across a suite of comprehensive evaluation metrics assessing the generation fidelity, diversity and analytical utility. We demonstrate how data augmentation using COSYNE helps to significantly improve downstream ML performance. We also show that COSYNE helps to drastically reduce time-to-value by being able to generate high fidelity simulated batches even when just a fraction of real data is available. As part of future work, a) COSYNE can leverage more advanced deep learning methods like transformers for sequence modeling and b) leverage more recent generative paradigms like diffusion models to better learn the spatio-temporal distributions.

References

[1] A. Alaa, B. Van Breugel, E. S. Saveliev, and M. van der Schaar. How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models. In *International Conference on Machine Learning*, pages 290–306. PMLR, 2022.

[2] M. K. Baowaly, C.-C. Lin, C.-L. Liu, and K.-T. Chen. Synthesizing electronic health records using improved generative adversarial networks. *Journal of the American Medical Informatics Association*, 26 (3):228–241, 2019.

[3] T. P. Carvalho, F. Soares, R. Vita, R. da Piedade Francisco, J. P. Basto, and S. G. S. Alcalá. A systematic literature review of machine learning methods applied to predictive maintenance. *Comput. Ind. Eng.*, 137, 2019.

[4] S. Chandra, M. Duvinage, P. Prakash, P. Davis, and S. Timmer. Supplementary material : Improving process yield through manufacturing digital twin using conditional synthetic data engine (cosyne). https://doi.org/10.5281/zenodo.13830291, 2024.

[5] P. Chen, T. Pedersen, B. Bak-Jensen, and Z. Chen. Arima-based time series model of stochastic wind power generation. *IEEE transactions on power systems*, 25(2):667–676, 2009.

[6] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[8] E. Choi, S. Biswal, B. Malin, J. Duke, W. F. Stewart, and J. Sun. Generating multi-label discrete patient records using generative adversarial networks. In F. Doshi-Velez, J. Fackler, D. Kale, R. Ranganath, B. Wallace, and J. Wiens, editors, *Proceedings of the 2nd Machine Learning for Healthcare Conference*, volume 68 of *Proceedings of Machine Learning Research*, pages 286–305. PMLR, 18–19 Aug 2017. URL https://proceedings.mlr.press/v68/choi17a.html.

[9] J. Deutsch and D. He. Using deep learning-based approach to predict remaining useful life of rotating components. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(1):11–20, 2018.

[10] D. Dowson and B. Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

[11] A. W. Edwards. Ra fischer, statistical methods for research workers, (1925). In *Landmark Writings in Western Mathematics 1640-1940*, pages 856–870. Elsevier, 2005.

[12] A. Efstratiadis, Y. G. Dialynas, S. Kozanis, and D. Koutsoyiannis. A multivariate stochastic model for the generation of synthetic time series at multiple time scales reproducing long-term persistence. *Environmental Modelling & Software*, 62:139–152, 2014.

[13] C. Esteban, S. L. Hyland, and G. Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[14] G. Forestier, F. Petitjean, H. A. Dau, G. I. Webb, and E. Keogh. Generating synthetic time series to augment sparse datasets. In *2017 IEEE international conference on data mining (ICDM)*, pages 865–870. IEEE, 2017.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[16] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[17] A. Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

[18] M. Kang, W. Shim, M. Cho, and J. Park. Rebooting acgan: Auxiliary classifier gans with stable training. *Advances in neural information processing systems*, 34:23505–23518, 2021.

[19] A. M. Lamb, A. G. ALIAS PARTH GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29, 2016.

[20] A. Malliaris. Wiener process. In *Time Series and Statistics*, pages 316–318. Springer, 1990.

[21] K. Medjaher, D. A. Tobon-Mejia, and N. Zerhouni. Remaining useful life estimation of critical components with application to bearings. *IEEE Transactions on Reliability*, 61(2):292–302, 2012.

[22] O. Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

[23] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier GANs. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/odena17a.html.

[24] A. Shamshad, M. Bawadi, W. W. Hussin, T. A. Majid, and S. Sanusi. First and second order markov chain models for synthetic generation of wind speed time series. *Energy*, 30(5):693–708, 2005.

[25] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[26] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks, 2014.

[27] P. W. Talbot, C. Rabiti, A. Alfonsi, C. Krome, M. R. Kunz, A. Epiney, C. Wang, and D. Mandelli. Correlated synthetic time series generation for energy system simulations using fourier and arma signal processing. *International Journal of Energy Research*, 44(10):8144–8155, 2020.

[28] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

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

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

[31] J. Yoon, D. Jarrett, and M. van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.

[32] Z. M. Çınar, A. Abdussalam Nuhu, Q. Zeeshan, O. Korhan, M. Asmael, and B. Safaei. Machine learning in predictive maintenance towards sustainable smart manufacturing in industry 4.0. *Sustainability*, 12(19): 2071–1050, 2020.