Generative Modeling for Healthcare Applications and Energy Demand Response with Normalizing Flows

Japjot Singh

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-162
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-162.html

May 20, 2022

Acknowledgement

Thank you to my advisor, Professor Costas Spanos, for opening the door for me to research opportunities at Berkeley and helping shape my work. Thank you to Hari Prasanna Das for introducing me to Normalizing Flows, collaborating closely with me on several projects over the last two years, and being a fantastic research mentor. Thank you to Lucas Spangher for mentoring me this semester and providing support in my work on energy demand response. Thank you to Professor Alberto Sangiovanni-Vincentelli for feedback and suggestions on this work.

Finally, thank you to my family and friends for your unparalleled support and unconditional love throughout my time at Berkeley--I feel incredibly lucky to have you all by my side.

Generative Modeling for Healthcare Applications and Energy Demand Response with Normalizing Flows

by Japjot Singh

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Costas J. Spanos
Costas J. Spanos (May 20, 2022 07:01 PDT)

Professor Costas J. Spanos
Research Advisor

May 20, 2022

(Date)

* * * * * * *

Alberto Sangiovanni-Vincentelli
Alberto Sangiovanni-Vincentelli (May 19, 2022 15:39 GMT+2)

Professor Alberto L. Sangiovanni-Vincentelli
Second Reader

May 19, 2022

(Date)

Generative Modeling for Healthcare Applications and Energy Demand Response with Normalizing Flows

Abstract

Generative Modeling for Healthcare Applications and Energy Demand Response with Normalizing Flows

by

Japjot Singh

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Costas J. Spanos, Chair

In the past decade, Machine Learning research has grown tremendously. The increased availability of data and powerful hardware has brought forward many applications in different industries. Generative modeling, specifically synthetic data generation, has made headlines with models capable of creating fake celebrity images and deep fakes. Normalizing Flows are one family of generative models with desirable qualities, including exact density estimation and inexpensive sampling. Unlike other generative modeling techniques like generative adversarial networks, variational autoencoders, and autoregressive models, Normalizing Flows show impressive results on both image and non-structured tabular data indicating their effectiveness in modeling complex distributions. Although still in relative infancy, they have shown promising results when used with other models or as a synthetic data source for separate downstream tasks.

This thesis explores applications in computer vision-based detection of COVID-19 and supervisory planning in reinforcement learning for energy demand response. Our work in the healthcare application presents a hybrid conditional generative model which decouples feature representations from input images to generate quality artificial samples in label scarce domains, which prove to be effective in several downstream tasks. We further investigate the flexibility of normalizing flow methods to capture energy price responses within a proprietary reinforcement learning environment and use this in a hybrid planning model scheme which in turn improves the learning and performance of a price controlling agent.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank some of the people who supported me throughout my academic journey. Thank you to my advisor, Professor Costas Spanos, for opening the door for me to research opportunities at Berkeley and helping shape my work. Thank you to Hari Prasanna Das for introducing me to Normalizing Flows, collaborating closely with me on several projects over the last two years, and being a fantastic research mentor. Thank you to Lucas Spangher for mentoring me this semester and providing support in my work on energy demand response. Thank you to Professor Alberto Sangiovanni-Vincentelli for feedback and suggestions on this work. Thank you to all of the outstanding Berkeley faculty I had the privilege to learn from and who helped me build a strong technical foundation and fostered my academic curiosity.

Finally, thank you to my family and friends for your unparalleled support and unconditional love throughout my time at Berkeley–I feel incredibly lucky to have you all by my side.

Chapter 1

Introduction

1.1 Background in Machine Learning

In recent years, the availability of large datasets combined with improved algorithms and an exponential growth in computation power has led to a surge of attention in Machine Learning (ML). State of the art ML has shown great success on tasks including classification, regression, and clustering, especially on complex and high-dimensional input data distributions. Breakthroughs in neural network sizes, architectures, and fitting procedures have transformed the use of ML and have shown superhuman abilities in difficult tasks (such as driving-cars [50], image classification [82], playing go [76]). As a result, ML has become a large part of many peoples' daily lives–for example: speech-recognition [18], fraud detection [5], email filtering [13], chatbots [60], search engines [8] and many more are all powered by machine learning algorithms.

Despite these advances, ML methods are often stymied by a lack of data availability in the real world, which hobbles one of science's most exciting new tools; unfortunately, these trends occur as the societal problems ML may address, such as pathogen-related global health and climate change, become worse and worse. A prominent branch of ML, generative modeling, has proven to be effective in capturing important statistical properties of the underlying data and using that to create synthetic samples. Artificially generated data is inexpensive compared to collecting large datasets, and is useful when privacy requirements limit data availability, if the data needed does not exist or is not available, and when enough training data is not available. Many industries and business functions benefit from using synthetic data, but in this work we will focus on two: healthcare for general pandemic responses (i.e. COVID-19) and energy (i.e. energy microgrids [24, 33] and energy demand response).

1.2 Pandemic Response

The COVID-19 pandemic created a public health crisis and continues to impact lives and healthcare systems worldwide. In the fight against this pandemic, a number of algorithms involving state-of-the-art machine learning techniques have been proposed. Data-based approaches have been used in a number of important tasks such as detection, mitigation, transmission modeling, decision making on restrictions etc. For example, computer vision-based detection of COVID-19 from chest computed tomography (CT) images has been proposed as a supportive screening tool for COVID-19 [28], along with the primary diagnostic test of transcription polymerase chain reaction (RT-PCR). This is beneficial since obtaining definitive RT-PCR test results may take a lot of time in critical situations[10]. Reinforcement learning based methods were also proposed to optimize mitigation policies that minimize the economic impact without overwhelming the hospital capacity [52].

The application of machine learning algorithms in healthcare depends upon ample availability of disease data along with their attributes and labels. At the beginning of a pandemic, data corresponding to the disease might be unavailable or sparse. Sparse data often have limited variation in several important factors relevant to disease detection such as age, and underlying medical conditions. Class imbalance is another issue faced by machine learning algorithms when pandemic-disease related data is limited. For example, at the onset of COVID-19, the number of CT scan images corresponding to COVID-19 was far less than that of other existing lung diseases (e.g. pneumonia). ML models fed with such class-imbalanced data could be biased and thus provide inaccurate results. Furthermore, the amount of data with correct labels among all available pandemic data might be minimal. This issue can arise because healthcare professionals and domain experts who are able to review and label the data are busy treating patients inflicted with the new disease, or also because of privacy concerns associated with medical data sharing.

Concurrently, after a new disease has been discovered, healthcare ML tools must rapidly adapt to the new disease in order to assist medical professionals in diagnosing and treating affected individuals as quickly as possible. A swift response is also necessary in the design of policy interventions based on insights from pandemic data. In addition to speed of response, another issue in development of machine learning algorithms for emerging pandemics is privacy [63, 17]. Development of solutions to pandemics at the scale of COVID-19 requires collaborative research which in turn presses the need for open-sourced healthcare data. But, even if healthcare organizations wish to release relevant data, they are often restricted in the amount of data to be released due to legal, privacy and other concerns. In this this work we present a novel conditional synthetic data generation method for augmenting COVID-19 CT-scan data and motivated by these results we explore the effectiveness of flow based models in a much more challenging reinforcement learning settings.

1.3 Energy Demand Response

As electrical grids decarbonize to assist in achieving climate goals, natural resources like wind and solar will replace non-rewnewable resources like fossil fuels. This change from an on-demand energy resource to a volatile one brings out an inconsistency between energy generation and demand. This paves the way for demand response, where customers will adjust their demand for energy resources to hours where generation is plentiful. Furthermore, with the recent advances in photovoltaic technology solar panels have become an efficient and reliable solution for customers to harvest solar energy for their own direct use and resale. These prosumers, in addition to large electric companies, are integral players in energy marketplace.

As with any marketplace, there is an opportunity for price-setters to buy and sell units at optimized prices to shift market demand and potentially generate profit. However, given the complexity, non-stationarity of demand response applying traditional optimization methods is methods is difficult. This is an appropriate setting for Reinforcement Learning (RL), an area of machine learning where an agent repeatedly takes an action, observes the result of this action in the system, and learns how to take actions which will maximize its notion of reward. Where traditional methods struggle with handling stochasticity in such systems, RL methods focus on balancing exploration and exploitation of optimal actions in light of uncertainty with dynamic programming techniques and deep neural networks.

In this work we consider prosumer aggregations which facilitate the trading of energy between participants in the aggregation, and balance the net load by purchasing from or selling to the utility. These prosumer aggregations can be formed for several different applications: a private entity can manage these aggregations for a fee or for a profit, and participants could cooperate their aggregations to maximize social welfare. Each aggregation controls the energy consumption and generation of each participant. Each prosumer will have an independent cost minimization objective, and will seek to optimize their time of use to meet their demand and maximize their own profits. The price of electricity directly influences the operation of these independent entities, and this strategy is denoted as **transactive control** [7]. It is easier for prosumers to respond to a day-ahead price as opposed to real time prices which may require predicting their load/generation schedule in advance. The aggregator is able to communicate prices one day-ahead to the participants, who will also process the utility prices and then schedule their daily operation to their objective. The aggregator's task is to design prices to reach its own objective (profit maximization, emissions reduction etc) while dealing with an uncertain environment. This uncertainty stems from several factors: the load and response of prosumers to energy prices is unknown to the aggregator, and the generation of electricity by prosumers is unpredictable and has inherent weather driven stochasticity.

1.4 Method

The purpose of this work is to experiment with the flexibility of flow based methods as a source of generative modeling.

We begin by presenting a hybrid model consisting of a conditional generative flow and a classifier for conditional synthetic data generation. The classifier decouples an input image's feature representation which is then fed through the flow to remove local noise from the underlying signal. We then generate synthetic data by perturbing local noise within the fixed underlying signal. We also propose a semi-supervised approach to generate samples in the case of label scarcity.

To further assess their effectiveness we investigate the performance of flows on non-structured tabular data in a proprietary reinforcement learning environment. We examine how offline training and supervisory planning can be leveraged to minimize data and learning costs of a price controller agent in a energy demand response context.

1.5 Outline of Thesis

This chapter provides a general overview of the thesis topic, goal, and approach. Chapter 2 covers some preliminaries, and related work. In Chapter 3 we describes the general methodologies for our approach and Chapter 4 details the experiments we ran and shows our results. A discussion of this work is provided in Chapter 5 followed by a conclusion in Chapter 6.

1.6 Novelty

To our knowledge, we present novel innovations in both of the methods we consider and we are the first to investigate applications of flow based methods in two divergent applications.

In healthcare, we present a novel conditional synthetic generative model which is effective in creating samples which lead to improved performance in a supervised task (detection of COVID-19). We also propose and experiment with different approaches to efficiently generate data under label scarce conditions, representing a significant methodological contribution.

In energy demand response, we are the first to try summarizing energy price responses using a continuous normalizing flow model. Methodologically, we are the first to use that specific generative model as a planning model to simulate steps in an RL framework.

Chapter 2

Background

2.1 Generative Modeling

This work focuses on applying a specific class of deep generative models to two different classes of learning problems: supervised learning and reinforcement learning. For the sake of this work, we will reiterate a conventional definition that we adhere to of **generative modeling**: the unsupervised ML task of learning underlying patterns of the input data in a model which can be used to artificially create new samples that plausibly could have been sampled from the input data [9]. In this background section we will provide an overview of the framework we are working with.

The increase in GPU technology over the last decade has led to developments of deeper generative models capable of creating fake celebrity images, and deep fakes. These applications will pose legal and ethical challenges but also promise new beneficial technologies. The potential for applications has brought considerable research interest to generative modeling in recent years.

Deep generative models are neural networks with millions of parameters and many hidden layers used to approximate complex, highly dimensional probability distributions. The shared goal of all generative models is to learn some unknown, and potentially intractable probability distribution ρ_0 from some number of independent and identically distributed samples. A successfully, trained DGM can be used to calculate the likelihood of a given sample $x \sim \rho_0$, and to create new samples resembling those from ρ_0.

Despite recent success, DGM suffers several key mathematical challenges [72]:

1. Generative model training is ill-posed from a information theoretic perspective: identifying a unique probability distribution from a finite number of samples is impossible.

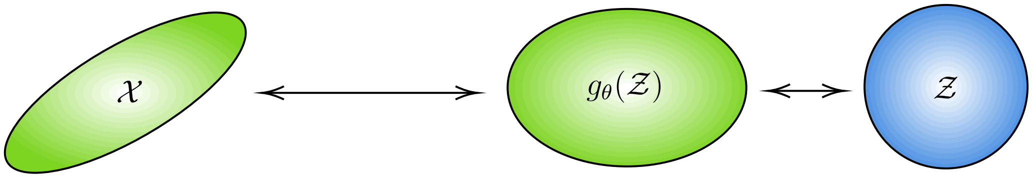

Figure 2.1: We train a generative model g_θ to transform samples from a simple distribution $\mathcal{Z}$ to $g_\theta(\mathcal{Z})$. The training objective is to make $g_\theta(\mathcal{Z})$ indistinguishable from $\mathcal{X}$.

This is why generative models are sensitive to network architecture, hyperparameters and training algorithms.

2. We need a way to test the quality of generator samples, specifically how close they resemble the source distribution $\mathcal{X}$. We can do this by either inverting the generator, or comparing the distribution of synthetic data $g_\theta(\mathcal{Z})$ to $\mathcal{X}$. Inverting a generator is not difficult if the generator is linear, but a linear neural network architecture will severely limit expressiveness and sample quality. Comparing distributions is not easy by any means either. They require us to conduct a two-sample test problem during training, which is difficult without any prior assumptions on $\mathcal{X}$ and $g_\theta(\mathcal{Z})$.

3. Most DGM approaches approximate intractable $\mathcal{X}$ by learning a transformation between $\mathcal{X}$ and some known distribution such as a Gaussian in a d-dimensional latent space. The choice of the latent space is critical to performance, but usually impossible to determine and is left as hyperparameter. If the latent space is too small, the generator will struggle to approximate the data and fit $\mathcal{X}$ poorly. If the latent space is too large we end up with a generator that is not injective which makes training difficult.

Mathematical formulation

Generative models try to learn a representation of intractable distribution $\mathcal{X}$ with support $\mathbb{R}^n$ where n is large and the distribution is complex. For example consider MNIST [61], each image is 28-by-28 so $n = 784$ representing each pixel in an image and $x \in \mathbb{R}^{784}$. We assume that we have access to a large but finite number of independent and identically distributed (i.i.d.) samples $x \sim \mathcal{X}$ referred to as our training data. Our goal is to learn a generator, parameterized by θ to map samples from a tractable distribution $\mathcal{Z} \in \mathbb{R}^d$ to $\mathbb{R}^n$ as illustrated in Figure 2.1. The key challenge is defining an objective function to quantify the difference between $\mathcal{X}$ and $g_\theta(\mathcal{Z})$. Once we have g we can generate new samples and compute the likelihoods of different samples.

Hand-designing a function to transform samples from a univariate Gaussian to images of celebrities is impossible and so we use deep neural networks (DNN) to parameterize g, hence the term deep generative models, so we denote the DNN generator g_θ and its weights by θ.

2.2 Finite Normalizing Flows

In normalizing flows we model the generator as a diffeomorphic and orientation-preserving function, in practice this is just the composition of invertible functions. This composition transforms the probability density from $\mathcal{X} \to \mathcal{Z}$ by repeatedly applying the change of variables formula and allows us to calculate the likelihood of a sample x as

$$p_{g_\theta}(x) = p_Z(g_\theta^{-1}(x))\left|\det\left(\frac{\partial g_\theta^{-1}(x)}{\partial x}\right)\right| \tag{2.1}$$

where invertibility requires that dim $(\mathcal{X}) =$ dim $(\mathcal{Z})$. Despite this restriction, flows can be used in conjunction with other approaches to work around this. The notation above may be slightly misleading since the flow function transforms the source distribution, $f : \mathcal{X} \to \mathcal{Z}$, and the generator is actually the inverse $g = f^{-1}$. During training the objective becomes to minimize the Kullback-Leibler (KL) divergence between $p_\mathcal{X}$ and p_{g_θ}. In practice this is intractable so we maximize the likelihood of samples from $\mathcal{X}$ under p_{g_θ} by minimizing the negative log-likelihood. As it turns out, the parameters θ which minimize this objective also minimize the KL divergence.

A finite normalizing flow is the full chain of invertible functions

$$g_\theta(z) = f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_K^{-1}(z) \tag{2.2}$$

where each layer f_i is invertible and has an easily computable Jacobian determinant.

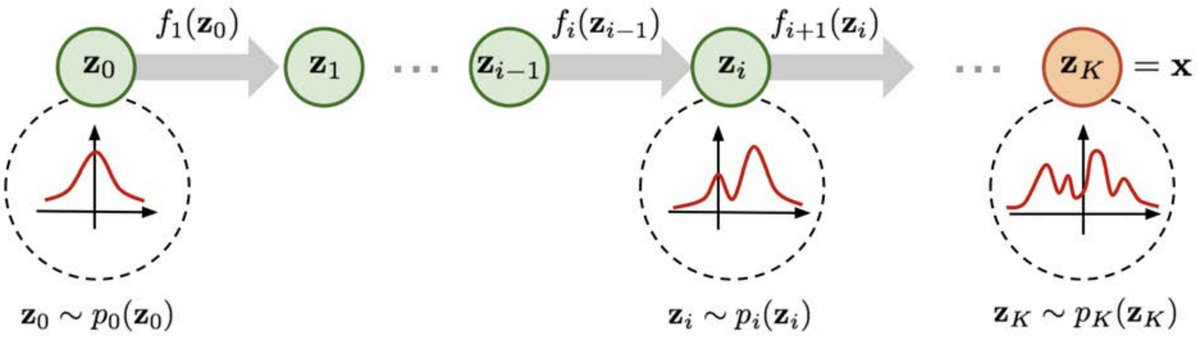
$f_1(\mathbf{z}_0)$
$\mathbf{z}_0$
$\mathbf{z}_1$
$\mathbf{z}_{i-1}$
$f_i(\mathbf{z}_{i-1})$
$\mathbf{z}_i$
$f_{i+1}(\mathbf{z}_i)$
$\mathbf{z}_K = \mathbf{x}$
$\mathbf{z}_0 \sim p_0(\mathbf{z}_0)$
$\mathbf{z}_i \sim p_i(\mathbf{z}_i)$
$\mathbf{z}_K \sim p_K(\mathbf{z}_K)$

Figure 2.2: Illustration of a normalizing flow model. In this figure we are transforming a simple distribution $p_0(z_0)$ to a complex one $p_K(z_K)$(Image source: [86])

We can perform maximum likelihood of a sample using

$$g_\theta^{-1}(x) = f_K \circ f_{K-1} \circ \cdots \circ f_1(x) \tag{2.3}$$

where $\log\det\nabla g_\theta^{-1}(x) = \sum_{j=1}^{K} \log\det\nabla f_j(y^{(j)})$, $y^{(j+1)} = f_j(y^{(j)})$ and $y^{(K+1)} = z = g_\theta^{-1}(x)$.

The tradeoff in finite normalizing flows is the design of the coupling layers f_i, between expressive transformations and tractable Jacobians. One approach is to use *affine coupling* layers. Each bijection $f_i : y^{(i-1)} \rightarrow y^i$ splits the dimensions in two parts:

$$\begin{aligned} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= x_{d+1:D} \odot \exp(s(x_{1:d}) + t(x_{1:d})) \end{aligned} \tag{2.4}$$

where $s(\cdot)$ and $t(\cdot)$ are scale and translation functions mapping $\mathbb{R}^d \rightarrow \mathbb{R}^{D-d}$, more details can be found in [19] and [20].

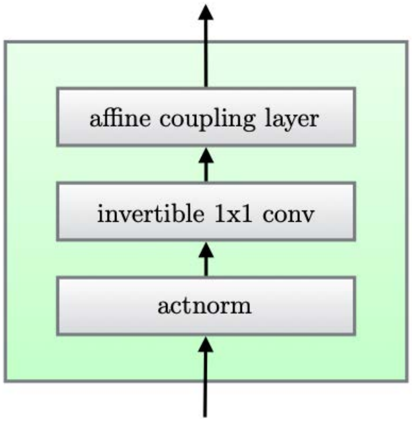

Figure 2.3: One step f_i of the Glow model. (Image source: [49])

The Glow model [49] extends on the previous models by replacing the permutation operation on channeling with 1x1 convolutions and uses the rest of the design same as RealNVP, details can be found in their paper.

2.3 Continuous Normalizing Flows

Continuous normalizing flows (CNF) remove the need to artificially design coupling layers and let the model learn those dynamics itself. CNFs approach the flow steps as an euler discretization of the continuous transformation $g_\theta^{-1} = f_K \circ \cdots \circ f_1(x)$. Taking this discretization in its limit defines a neural ODE characterizing the continuous dynamics of the hidden units. Given the parametrized ODE $v_\theta(y(t), t)$, we solve the initial value problem

$$\begin{aligned} y(t_0) &= z_0 \\ \frac{\partial y(t)}{\partial t} &= v_\theta(y(t), t) \end{aligned} \tag{2.5}$$

where v_θ is a neural network. The change in log-density under this model follows a second differential equation called *instantaneous change of variables* formula [11]:

$$\frac{\partial \log p(y(t))}{\partial t} = -\mathrm{Tr}\Big(\frac{\partial v}{\partial y(t)}\Big) \tag{2.6}$$

and we can compute the total change in log-density by integrating:

$$\log p(y(t_1)) = \log p(y(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\Big(\frac{\partial v}{\partial y(t)} dt\Big) \tag{2.7}$$

$$\begin{bmatrix} y(x,t) \\ l(x,t) \end{bmatrix} = \int_{t_1}^{t_0} \begin{bmatrix} v_\theta(y(x,t),t) \\ -\mathrm{Tr}(\frac{\partial v}{\partial y(t)}) \end{bmatrix} dt \tag{2.8}$$

with initial values

$$\begin{bmatrix} y(x,0) \\ l(x,0) \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix} \tag{2.9}$$

for some $t \in [0, T]$, $x \in R^d$ and $l(x,T) = \log \det \nabla y(x,T)$. The first component maps points from x to $g_\theta^{-1}(x) = y(x,T)$ subject to the dynamics v. The second component is derived from the instantaneous change of variables formula. The dynamics are trained by minimizing negative likelihood, details can be found in [65].

2.4 Related Works

In healthcare, synthetic data generation has been proposed to expand the diversity and amount of the existing training data, often to improve the robustness of machine learning models. [24] propose a generative adversarial network (GAN)-based synthetic data generator to enhance the diversity and the amount of skin lesion images. [51] synthesize pathology images for cancer with realistic out-of-focus characteristics to evaluate general pathology images for focus quality issues. [32] propose synthetic generation to produce high-resolution artificial radiographs. In combating COVID-19, [6] propose a method of strengthening the COVID-19 forecasts from compartmental models by using short-term predictions from a curve fitting approach as synthetic data. Similarly, [84] and [45] present a conditional

GAN-based generator for synthetic chest X-ray/CT scan data generation and augmentation for robust COVID-19 detection. These prior works do not focus on the case where data with proper labels might be unavailable or sparsely available, whereas we tackle this challenge using a semi-supervised approach. We also show the robustness achieved using our model via experiments with several bootstrapping methods.

There has been considerable work studying demand response interactions in energy demand response [21, 1, 2, 29]. Works in this area investigate energy market equilibrium behavior and focus on consumer response to prices. In contrast, we focus on controlling pricing to learn and manipulate both consumer responses and market behavior. There is prior work on coordinating pricing strategies of both appliances and buildings to achieve demand profiles [56, 78, 15, 47], but these strategies achieve dynamic energy management using direct model predictive control. Instead, we achieve dynamic pricing using RL, which is still relatively new [46, 55, 77, 3, 43]. Specifically, the use of supervisory planning models in online learning, especially in an energy demand response setting, is relatively new [42, 44]. One of the most important components of these planning models is the ability to capture the underlying stochasticity of demand in our environment. Competitions such as The Global Energy Forecasting Competition (GEFCom) have showcased numerous successful forecasting techniques including quantile regression, random forests, autoregressive methods, and neural networks [36, 38, 37] . Unfortunately, these methods are not as successful when extended to multivariate distributions. Our use of continuous normalizing flows to summarize these distributions in conjunction with a planning model contributes to RL literature.

In conditional generation, a hybrid flow and a GAN-based model have been proposed in CAGlow [54]. In general, GAN-based methods are known to be hard to train [73] and do not provide a latent embedding suitable for feature manipulations [49]. In contrast, we proposed a conditional generation method with efficient decoupling of the conditional information and local noise over an embedding space, along with a flow based generator, which recently has proved efficient in synthetic data generation [35, 16]. We compared results for our proposed method over CAGlow and ACGAN for synthetic COVID CT scan generation, and showed improved results.

Decoupling of global and local representation for synthetic generation has been proposed in [58], where the global information is decoupled using a Variational AutoEncoder (VAE) [48]. For conditional synthetic generation, it is necessary that the feature representations salient to the given conditions (COVID/Non-COVID) are decoupled from local noise, which is not guaranteed while extracting the same using a VAE. By employing a classifier network for the same, we ensure the relevant conditional information is not lost in the local noise.

Semi-supervised learning approaches to enhance classification models have been prominent in domain adaptation tasks, where knowledge about the labels is generally unavailable in the target domain except for a few samples. A number of domain adaptation models, such as FADA [62, 81, 89, 87] etc. employ few-shot learning approach, leveraging the few labeled data available to make the model efficient. In healthcare, semi-supervised learning approaches have been used for skin disease identification from limited labeled samples in [59], to enhance X-ray classification in [68] and in COVID-19 detection from scarce chest X-ray image data in [41]. We propose the use of semi-supervised learning in the space of synthetic data generation, to adapt our proposed generative model to label scarce scenarios, common at the onset of a pandemic.

Chapter 3

Methodology

Deep convolutional neural networks (CNN) have proven to be remarkably effective on classification tasks; however, these networks rely on a large amount of data to prevent overfitting. Overfitting is a phenomenon that occurs when a statistical model learns a high variance function to fit the training data perfectly but fails to accurately classify data the model has not seen before. Furthermore, many applications of CNNs are held back by a lack of access to big data, including those in medical fields and the energy sector. As a result, there is currently a significant effort to create and improve Data Augmentation techniques that can enhance the size and quality of datasets used by CNNs. We can primarily break down these data augmentation techniques into three categories: geometric augmentation, photometric augmentation, and entropy-inducing augmentations [75]. Geometric transformations include rotation, flipping, and cropping. Photometric transformations include sharpening, color casting, jittering, and edge enhancements. Unlike geometric and photometric augmentations, entropy-inducing augmentations are not transformations. Specifically, these augmentations require adding datapoints that do not already exist in the data. For example, if we in-paint accessories (i.e., earrings, jewelry, hat) onto an individual and include that as a new sample, it would be considered an entropy-inducing augmentation. Similarly, if we were working with an image generation model and augmented our original dataset with synthetically generated images from a generative adversarial network (GAN) [25], those images would be entropy-inducing since they are adding new and unseen samples to our data. This work will use normalizing flows as our deep generative model of choice for image generation.

Geometric and Photometric transformations are sufficient for augmentation when the raw dataset is already able to provide decent performance. These augmentations add an inductive bias which encourages the network to focus on patterns in local image structure for classification. Entropy-inducing augmentations are necessary when working with imbalanced datasets or label scarce datasets. An imbalanced dataset does not provide enough information to learn an effective deep network. The objective will encourage the network to focus on classifying the majority class correctly while neglecting the minority class, and this is reflected in poor f1-scores. In label scarce domains we are forced to choose between lower classification accuracy and a smaller dataset since samples with no labels cannot be used for training in a supervised classification model. We seek to resolve both of these problems with Normalizing Flows.

3.1 Augmenting medical image-data

In this section, we present a novel conditional synthetic data-generation method to augment the available pandemic data of interest. Our proposed method can also help organizations release synthetic versions of their actual data with similar behavior in a privacy-preserving manner. At the onset of a pandemic, when the availability of disease data is limited, our proposed model learns the distribution of available limited data and then generates conditional synthetic data that can be added to the existing data in order to improve the performance of machine learning algorithms. To tackle the challenge of label scarcity, we propose semi-supervised learning methods to leverage the small amount of labeled data and still generate qualitative synthetic samples. Our methods can enable healthcare ML tools to adapt to a pandemic rapidly.

We apply this method to generate conditional CT scan images corresponding to COVID cases (Fig. 3.1) and conduct qualitative and quantitative tests to ensure that our model generates high-fidelity samples and is able to preserve the features corresponding to the condition (COVID/Non-COVID) in synthetic samples. As a downstream use of conditional synthetic data, we improve the performance of COVID-19 detectors based on CT scan data via synthetic data augmentation. Our results show that the proposed model is able to generate synthetic data that mimic the real data, and the generated samples can indeed be augmented with existing data in order to improve COVID-19 detection efficiency.

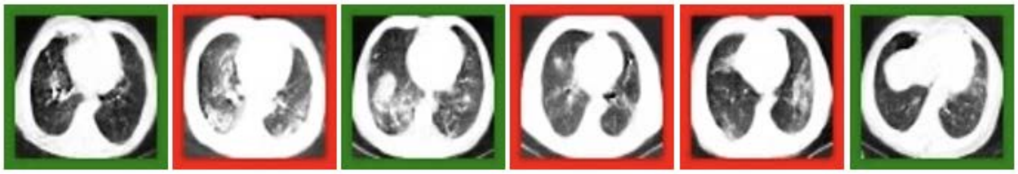

Figure 3.1: Synthetic CT scans generated by our proposed model, with Non-COVID (normal and pneumonia cases, images with green border)/ COVID (images with red border) as the condition.

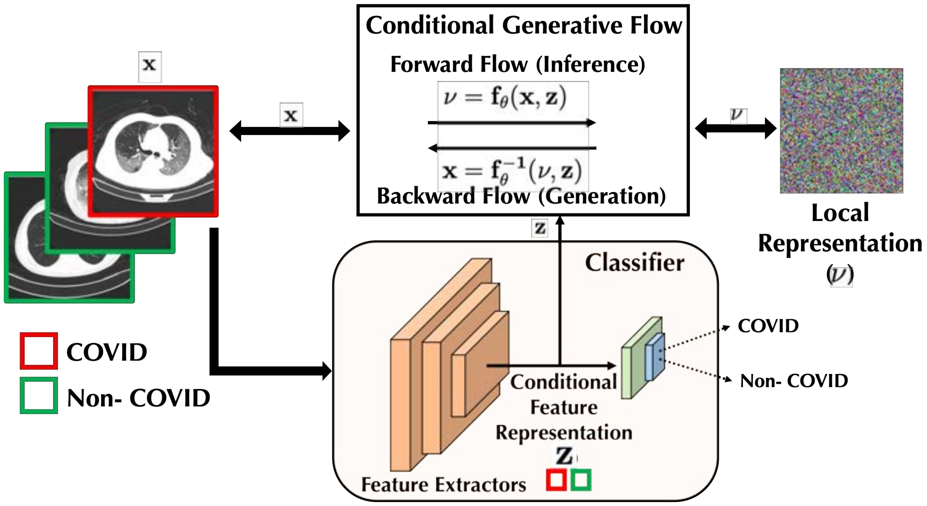
x
x
Conditional Generative Flow
Forward Flow (Inference)
$\nu = \mathbf{f}_\theta(\mathbf{x}, \mathbf{z})$
$\mathbf{x} = \mathbf{f}_\theta^{-1}(\nu, \mathbf{z})$
Backward Flow (Generation)
ν
Local Representation (ν)
z
Classifier
COVID
Non- COVID
Conditional Feature Representation
Z
Feature Extractors
COVID
Non- COVID

Figure 3.2: Illustration of the proposed conditional synthetic generation. (Best viewed in color)

Inference Phase	Generation Phase
1. **(Classifier)** Train the COVID and Non-COVID classifier. 2. **(Flow)** For each input sample x, 2.1 Feed x to the classifier and extract the conditional feature representation z from its penultimate layer. 2.2 Get the local representation as $\nu = f_\theta(x, z)$ 2.3 Train the flow model with maximum-likelihood.	1. **(Classifier)** Corresponding to an input sample x, find its conditional feature representation z using the trained classifier. 2. **(Flow)** Sample a local representation $\tilde{\nu} \sim \mathcal{N}(0, I)$. 3. **(Flow)** Get a synthetic sample $\tilde{x} = f_\theta^{-1}(\tilde{\nu}, z)$.

Table 3.1: Summary of steps for conditional inference and generation

We present a hybrid model consisting of a conditional generative flow and a classifier for conditional synthetic generation. We also introduce a semi-supervised approach, to generate conditional synthetic samples when a few samples out of the whole dataset are labeled.

COVID and Non-COVID Classifier

Our model is characterized by the efficient decoupling of feature representations corresponding to the condition and the local noise. Suppose we have N samples $\mathbf{X}$ with labels Y, with 2 possible classes, COVID/Non-COVID. We first train a classifier C (consisting of a feature extractor network denoted by $g(\cdot)$, and a final fully-connected and softmax layer, denoted by $h(\cdot)$, i.e. $C(x) = h(g(x))$) to classify the input sample (which in our case are CT Scans) and associated labels as COVID and Non-COVID. Mathematically, this step solves the following minimization with backpropagation:

$$\min_{C} \mathcal{L}_C(\mathbf{X}, Y) = -\mathbb{E}_{(x,y)\sim(\mathbf{X},Y)} \sum_{l=1}^{2} \left[\mathbb{I}_{[l=y]} \log C(x))\right] \tag{3.1}$$

By virtue of the training process, the classifier learns to discard local information and preserve the features necessary for classification (conditional information) towards the downstream layers. Once the classifier is trained, we freeze its parameters and use it to extract the conditional (COVID/Non-COVID) feature representation $z = g(x)$ (as a vector without spatial characteristics) at the output of the feature extractor network for input image x. The dimension of z is chosen such that $\dim(z) << \dim(x)$.

Conditional Generative Flow

During the training phase for the flow model, the conditional feature representation z is fed to the conditional generative flow. The flow model is trained using maximum-likelihood, transforming x to its local representation ν, i.e.

$$f_\theta(x, z) = \nu \sim \mathcal{N}(0, I) \tag{3.2}$$

with ν having the same dimension as x by the inherent design of flow models. We use the method introduced by [58] to incorporate the conditional input z in flow model. Coupling layers in affine flow models have scale ($s(\cdot)$) and shift ($b(\cdot)$) networks [20, 14], which are fed with inputs after splitting, and their outputs are concatenated before passing on to the next layer. We incorporate the conditional information z in the scale and shift networks. Mathematically, (with x as the input, D as input dimension, d as the split size,and y as output of the layer),

$$\begin{aligned} x_{1:d}, x_{d+1:D} &= \text{split}(x) \\ y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= s(x_{1:d}, z) \odot x_{d+1:D} + b(x_{1:d}, z) \\ y &= \text{concat}(y_{1:d}, y_{d+1:D}) \end{aligned} \tag{3.3}$$

Since flow models are bijective mappings, the exact x can be reconstructed by the inverse flow with z and ν as inputs. During the generation phase, for an input sample x, we compute the conditional feature representation z. Keeping the conditional feature representation the same, we sample a new local representation $\tilde{\nu}$, and generate a conditional synthetic sample $\tilde{x}$, i.e.

$$\tilde{\nu} \in \mathcal{N}(0, I), \quad \tilde{x} = f_\theta^{-1}(\tilde{\nu}, z) \tag{3.4}$$

Here, $\tilde{x}$ has the same conditional (COVID/Non-COVID) features as x , but has a different local representation. An illustration of the proposed model is provided in Fig. 3.2 and the steps for the inference and generation phases are summarized in Table 3.1.

Semi-supervised Learning for Conditional Synthetic Generation under Label Scarcity

In reality, often a small amount of the already limited pandemic data available are labeled. Consider the case when only a few of the datapoints are labeled, denoted by $\{\mathbf{X}^l, Y^l\}$. The rest of the data (unlabeled) is denoted by $\mathbf{X}^u$. To generate conditional synthetic samples under such label scarce situations, we propose a semi-supervised method to modify the classifier design process, in order to effectively decouple the feature representations corresponding to the conditions.

We first design a label learning algorithm to assign presumptive labels $\tilde{Y}^l$ to the unlabeled samples $\mathbf{X}^u$. Assuming k_i labeled samples are available for class i, we train the classifier network using the labeled samples only and compute in the embedded (z) space (1) the centroid vector c_i for each class and (2) a similarity metric between each unlabeled target sample $x^u \in \mathbf{X}^u$ and the specific centroid. Depending on the dimension of the transformed feature space, this similarity metric can simply be a Gaussian kernel to capture local similarity [83], or the inverse of Wasserstein distance [74] for better generalization with complex networks.

Ideally, the semi-supervised scheme should be able to (1) identify the correct labels of unlabeled target samples, and (2) update the classifier with the additional information. We establish an alternating approach that recursively performs (1) fixing the feature mapping g and propagating presumptive labels using a greedy assignment, i.e., an unlabeled sample is presumed to have the same label to its closest centroid, and (2) updating the feature mapping (the classifier) as supervised learning by treating the presumptive labels as true labels.

The proposed greedy propagation, intuitively simple and practically easy to implement, in fact, has theoretical guarantees since the entropy objective is approximately submodular when the feature mapping is fixed. Please refer to [88] for a detailed theoretical analysis. The above is conducted alternately until the convergence of the feature mapping and presumptive label assignment. In practice, the convergence is usually achieved in a few iterations. Once the classifier has been trained with this semi-supervised approach, the conditional generative flow training is performed as specified before in the conditional generation section.

3.2 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning method where an agent is trained in an environment where it learns to take actions that will maximize some reward. Formally, RL is defined as an MDP made up by the tuple: $(\mathcal{S}, \mathcal{A}, R, \gamma)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, R is the reward, and γ is a discount factor. The agent's functioning in the MDP is modeled as a discrete-time stochastic control process [22] where the agent interacts within its environment at $s_0 \in \mathcal{S}$, gathering an initial observation $\omega_0 \in \Omega$. At each timestep t the agent chooses an action $a_t \in \mathcal{A}$ according to a policy function, which maps $\mathcal{S} \longrightarrow \mathcal{A}$. As a consequence of actions, the agent receives a reward r_t, and the environment transitions to s_{t+1}. The agent seeks to maximize the long term discounted sum of rewards; i.e. maximize J where:

$$J = \sum_{t=0}^{\infty} \gamma^t R(s, a) \tag{3.5}$$

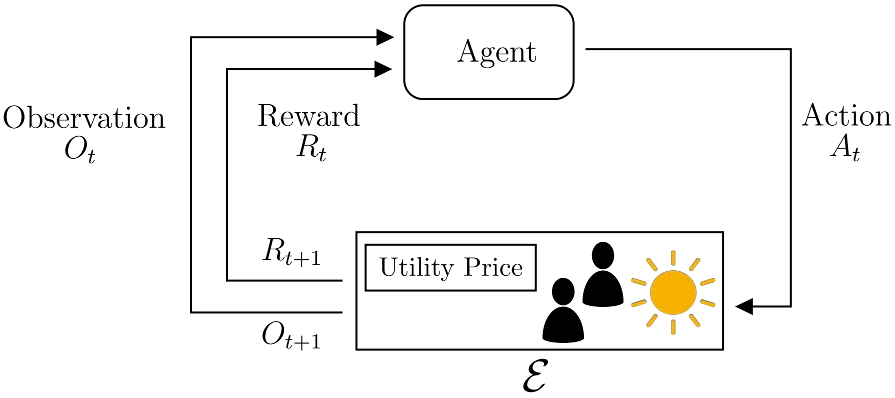

Figure 3.3: Agent-environment interaction in the microgrid environment $\mathcal{E}$. At each timestep, the agent takes an action A_t and obtains a reward R_{t+1}. The state of the environment transitions to S_{t+1} but since the system is partially observed and the agent only sees O_{t+1}.

The Microgrid Environment

As illustrated in Figure 3.3, the microgrid environment $\mathcal{E}$, simulates energy demand dynamics between an energy supplier, a middleman, and prosumers. At each time-step the agent selects an action $a_t = (a_t^{buy}, a_t^{sell})$ from the continuous 48-dimensional action space $\mathcal{A}$. We define a^{buy} as the price the prosumer will pay to the agent for 1 Kilowatt-hour (kWh) of energy, and a^{sell} as the price the agent will pay to the prosumer for 1 (kWh) of energy. This action is passed to the microgrid environment and modifies its internal state yielding a prosumer response $p_t \in \mathbb{R}^{24}$. In general $\mathcal{E}$ is stochastic. Each time-step represents one full day, and at the start of each timestep the external grid (large energy provider i.e. PG&E) outputs $u_t = (u_t^{buy}, u_t^{sell}) \in \mathbb{R}^{48}$. These external grid prices are determined from historical data. The environments internal state is not observed by the agent; instead the agent observes $o_t \in \mathbb{R}^{72}$ comprised of p_{t-1}, u_t, and a noisy reading of the total energy generated by prosumers on day t. The agent also receives a reward r_t representing its net cashflow.

The prosumer response $p_t(h)$ is difference between its electricity demand and generation at hour h. The prosumer will purchase $p_t(h)$(kWh) of energy if the value is positive, and sell $|p_t(h)|$ if the value is less than 0. The prosumer trade with the agent only if it provides a strictly competitive rate, i.e. $a_t^{buy}(h) < u_t^{buy}(h)$ or $a_t^{sell}(h) > u_t^{sell}(h)$. At each timestep the prosumer will run a convex linear optimization to minimize its total cost of energy for the day, thus we can define demand response as some unknown function:

$$f(s_t, a_t) = p_t \tag{3.6}$$

where $s_t \in S$ is the internal state of the grid.

We can configure $\mathcal{E}$ by specifying the number of participating prosumers, and battery sizes which define the energy generation capacity of each prosumer. We define the set of prosumers as $\mathcal{P}$, and denote prosumer participant j as p^j.

The microgrid requires the conservation of energy. The agent aims to insert itself into this equilibrium and make a profit by serving as the middleman between multiple prosumers and the external grid. To sustain this equilibrium, the agent must honor a_t, so it cannot default on its prices. This means if $a_t^{buy}(h) < u_t^{buy}(h)$ and the net demand response from prosumers $\sum_{j\in\mathcal{P}} p_t^j(h)$ is positive, then the agent is forced purchase this surplus from the external grid at rate $u_t^{sell}(h)$. Alternatively if $a_t^{sell}(h) > u_t^{buy}(h)$ and $\sum_{j\in\mathcal{P}} p_t^j(h) < 0$, then the agent has extra Kilowatt-hours which it will sell to the utility company at $u_t^{buy}(h)$ — the agent cannot save energy to use at later hours or timesteps.

Since the agent does not observe the internal state of our system, the task is partially observed, and many environment states are aliased, i.e., it is impossible to understand p_t from only o_t. The goal of the agent is to interact with $\mathcal{E}$ by selecting actions that will maximize reward by maximizing its net cashflow. Our time-horizon $T = 1$, since we are interested in a single-day feedback loop. This means our episode length is one day, and the agent receives its reward in dollars for that day.

Cashflow Reward

The agent's net cashflow, can be broken down by handling its interactions with the external grid and prosumers separately. From the discussion of energy equilibrium, we define the cashflow between the agent and the external grid as money to utility. If we define the net prosumer energy demand as $P_t(h) = \sum_{j \in \mathcal{P}} p_t^j(h)$ and $P_t(h) = 0$ when $a_t(h)$ is not a strictly competitive rate, then at timestep t we calculate the net cashflow to the utility from the agent:

$$\text{money to utility} = \sum_{h=1}^{24} \mathbb{1}_{P_t(h)>0}(P_t(h)a_t^{buy}(h)) + (1 - \mathbb{1}_{P_t(h)>0})(P_t(h)a_t^{sell}(h)) \tag{3.7}$$

where $P_t(h)a_t^{buy}(h)$ is a positive cashflow for the external grid (utility), and $P_t(h)a_t^{sell}(h)$ is a negative cashflow so the net cashflow to the utility is just the sum of these two terms. Meanwhile, we define the cashflow between the agent and the prosumers as money from prosumers. This interaction happens after the utility interaction to ensure the energy is conserved in our system and the agent has fulfilled its commitments. We calculate the net cashflow to the agent from the prosumers at timestep t:

$$\text{money from prosumers} = \sum_{j \in \mathcal{P}} \sum_{h=1}^{24} \mathbb{1}_{p_t^j(h)>0}(p_t^j(h)a_t^{buy}(h)) + (1 - \mathbb{1}_{p_t^j(h)>0})(p_t^j(h)a_t^{sell}(h)) \tag{3.8}$$

where $p_t^j(h)a_t^{buy}(h)$ is a positive cashflow and $p_t^j(h)a_t^{sell}(h)$ is a negative cashflow, so the net cashflow to the agent is the sum of these two terms. There is an arbitrage opportunity if $a_t^{sell}(h) > a_t^{buy}(h)$. Any prosumer would be able to drive money from prosumers to $-\infty$. However, the prosumer's demand response optimizes energy cost and will not take advantage of this situation, so we add a large penalty for actions that violate this inequality.

The RL agent's reward at timestep t as $r_t =$ (money from prosumers - money to utility):

$$\begin{aligned} r_t = &\sum_{j\in\mathcal{P}}\sum_{h=1}^{24} \mathbb{1}_{p_t^j(h)>0}(p_t^j(h)a_t^{buy}(h)) + (1-\mathbb{1}_{p_t^j(h)>0})(p_t^j(h)a_t^{sell}(h)) \\ &-\Big(\sum_{h=1}^{24}\mathbb{1}_{P_t(h)>0}(P_t(h)a_t^{buy}(h)) + (1-\mathbb{1}_{P_t(h)>0})(P_t(h)a_t^{sell}(h))\Big) \end{aligned} \tag{3.9}$$

Thus the goal of our agent is to maximize its profit margin. We also note that our reward is in units of USD. The dynamics of this profit margin are driven by the relationship between $a_t^{buy} - a_t^{sell}$, u_t, and p_t^j.

MDP formulation

After prices are posted the environment transitions from state s_t to s_{t+1} according to the transition function $\mathbb{P}(s_{t+1}|s_t, a_t)$. In MDP terminology, our RL agent is learning is a feasible policy $\pi(a_t|s_t)$ which specifies a distribution over prices $a_t \in \mathcal{A}$. When we repeatedly apply π across time we generate a state-action trajectory (or rollout) $(s_1, a_2, s_2, a_2, \ldots, s_T, a_T)$, where $\mathbb{P}_\pi(s_1, a_1, s_2, a_2, \ldots, s_T, a_T) = \mathbb{P}(s_1)\prod_{t=1}^{T}\pi(p_t|s_t)\mathbb{P}(s_{t+1}|s_t, p_t)$. The task of building a profitable agent is formulated as finding a policy that maximizes the discounted expected reward:

$$\max_{\pi\in\Pi}\mathbb{E}\Big[\sum_{t=1}^{T}\gamma^{t-1}r(s_t, a_t)|s_1\Big] \tag{3.10}$$

Where s_1 is our initial state and the expectation is maximized with respect to the trajectory induced by π and subject to environment transition dynamics.

This MDP cannot be solved directly since we do not know the state transition dynamics $\mathbb{P}(s_{t+1}|s_t, a_t)$, nor the demand response function $f : \mathcal{S}\times\mathcal{A}\to P$. We can leverage deep RL to tackle these issues.

Model-free RL

Model-based RL has been used extensively in energy management and grid control applications. The unknown functions are learned from offline data before optimizing the policy in these contexts. These models, which learn from the offline models, are referred to as *planning models* [31] in reinforcement learning because the agent spends time interacting with the planning model instead of the ”real world” environment.

Model-free RL skips the planning model by learning directly from the real-world environment to optimize the policy. At the core of this approach is the exploration-exploitation tradeoff. Exploration is the process of trying actions to understand the environment's response and reach more of the state space to understand the reward surface better, not just to achieve the highest reward. An example exploration strategy is to sample actions within a neighborhood around the agent's chosen price. Meanwhile, exploitation is strictly choosing actions to optimize short-term rewards. Since model-free RL does not contain a planning model, we rely on the policy to implicitly encode knowledge of the state space, so striking the right balance of exploration and exploitation is critical. The amount of data needed to learn a good pricing policy that maximizes reward translates to the number of days the agent incurs financial liability in the real world. This financial liability during the learning phase could make the model-free price controller economically not viable. This cost motivates the need for a nonstandard approach that is more favorable for deployment.

We propose a deep RL approach that uses a planning model to reduce the learning cost of model-free RL. We will use a model-free RL template that can effectively learn from a limited amount of real-world data and incurs a minimal learning cost.

Model-free deep RL approach

Policy Gradient and Value iteration are the two main RL techniques for model-free learning. We will focus on policy gradient and more specifically actor-critic architectures. Actor-critic methods have an actor-network which selects actions that are processed by a separate critic network that estimates the long-term value of the actions and nudges the policy accordingly. Proximal Policy Optimization (PPO) is a state-of-the-art actor-critic architecture that updates the policy by maximizing the PPO-Clip objective (taken from [67]):

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \min\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t))\Big) \tag{3.11}$$

where $\mathcal{D}_k = \{\tau_i\}$ is a set of trajectories collected by running policy $\pi_k = \pi(\theta_k)$ in the environment, $\hat{R}_t$ is the reward-to-go, and A_t is any method of advantage estimation based on V_{ϕ_k} where

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \Big(V_\phi(s_t) - \hat{R}_t\Big)^2 \tag{3.12}$$

We seek to explore if an RL agent can preemptively estimate the most profitable demand response price using historical data and implicitly predict causal factors. We investigate if we can pre-train an agent in simulations that can adapt to real-world data and whether or not this is more effective than employing a softer guardrails approach which includes incorporating offline data into online training.

We use the RLLib implementations of PPO; with its default neural network architecture, we provide training hyperparameters in Appendix A. The reward for the price-setting agent is the net cashflow defined in Equation 3.9.

3.3 Offline RL and Synthetic Data

In online reinforcement learning, the policy at update k, denoted π_k, is updated with data rollout(s) collected using π_k. In off-policy reinforcement learning, the agent accumulates its experience into a data buffer (replay buffer) $\mathcal{D}$ so that the buffer contains samples from $\pi_0, \pi_1, \ldots, \pi_k$ and all of this data is used to update the policy. In offline reinforcement learning, some unknown behavioral policy π_β is used to collect $\mathcal{D}$. Once $\mathcal{D}$ is collected, it is not altered, and the policy is trained without interacting with the MDP, and the policy is only deployed after being fully trained. Figure 3.4 illustrates the relationship between these different approaches.

In this section, we focus on *data-driven* online reinforcement learning. Specifically, we explore how we can leverage off-policy data to limit data-cost and learning-cost of our agent in an online setting. Data-cost is the cost of acquiring the rollout data, and learning-cost is the financial cost accumulated with an agent that loses money during rollout, where net cashflow is defined in Equation 3.9. We take this approach since the rollout data in $\mathcal{D}$ is not directly used to learn the policy π, unlike with off-policy algorithms like Soft Actor-Critic (SAC) [30]. So we use $\mathcal{D}$ to learn an approximation of the underlying transition function in the environment. This allows us to use the offline data in a supervised planning model.

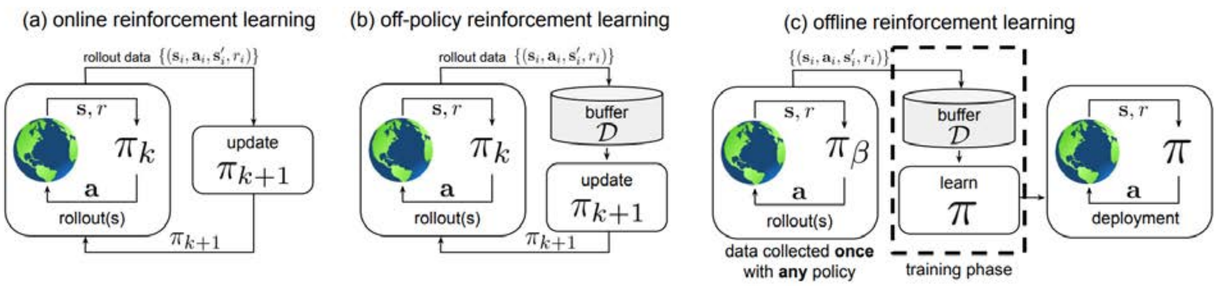
(a) online reinforcement learning
rollout data $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$
$\mathbf{s}, r$
π_k
$\mathbf{a}$
rollout(s)
update
π_{k+1}
π_{k+1}
(b) off-policy reinforcement learning
rollout data $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$
$\mathbf{s}, r$
π_k
$\mathbf{a}$
rollout(s)
buffer
$\mathcal{D}$
update
π_{k+1}
π_{k+1}
(c) offline reinforcement learning
$\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$
$\mathbf{s}, r$
π_β
$\mathbf{a}$
rollout(s)
data collected **once** with **any** policy
buffer
$\mathcal{D}$
learn
π
training phase
$\mathbf{s}, r$
π
$\mathbf{a}$
deployment

Figure 3.4: Illustration of classic online reinforcement learning (a), classic off-policy reinforcement learning (b), and classic offline reinforcement learning. (Image source: [53])

3.4 Synthetic Data Generation

Energy demand response datasets are expensive. A single sample in our dataset is an entire day of demand response interaction. Furthermore, the external grid prices u_t and prosumer responses for the first ten prosumer participants are based on real-world data. Experiments with offline RL done in this environment require upwards of 70 years of data [44], and in the real world, this can be an infeasible constraint. If we could get access to more data, we would be able to explore offline RL algorithms without incurring a high data cost. In the classical offline reinforcement learning settings, we store and use data collected from some policy π_β. We propose applying generative methods to augment the data buffer $\mathcal{D}$ in offline RL problems, and in the micro-grid environment, we can incorporate this data directly into a planning model. We will use $\pi_{\beta^\star}$ to denote the policy induced by synthetically generated data. Our goal will be to generate data that resembles real rollout data. This will reduce our data cost and significantly improve the performance of agents trained with *data-driven* methods. In this section, we will distinguish π_β and $\pi_{\beta^\star}$, and in practice, π_β can come from any policy. Our goal is to synthesize data inducing some $\pi_{\beta^\star}$ which spans the agent's action space, and then we can use this data to train a planning model which will allow the agent to learn a better policy and converge to that policy faster. In the remaining sections we handle π_β and $\pi_{\beta^\star}$ interchangeably and refer to an arbitrary behavior policy for offline training as π_β.

Continuous Normalizing Flows

The synthetic generation problem statement is to synthesize tabular data which resembles rollouts from π_β. The input data is 73-dimensional: $a_t \in \mathbb{R}^{48}$, $p_t \in \mathbb{R}^{24}$, and day $t \in [0, 364]$. We experimented with including the utility information and building information, which indicate the usage behavior of the specific prosumer who generates response p_t, but we prefer to leave accessible state variables out of the data since they add to the number of columns, greatly increasing the sparsity of our data and are not the most important covariates for the underlying system.

We use continuous normalizing flows because finite flow models fail to perform well on non-image datasets. This is because the choice of permutation heavily influences finite flows in their coupling layers. This is fine with image data since we can incorporate prior knowledge about image structure, like the importance of neighboring pixels and splitting across channels. There is no such intuition with tabular data, so we do not know how to come up with sensible partitions of these features. Free-Form Jacobian of Reversible Dynamics (FFJORD) [26] completely sidesteps this issue by deferring the decision of how to permute features to the network parameters. This leads to good performance, and in fact, FFJORD consistently outperforms competing reversible models. Furthermore, on tabular datasets, the performance of FFJORD is on par with state-of-the-art autoregressive methods (MADE [23], MAF [66], TAN [64], MAF-DDSF [40]) all of which cannot be efficiently sampled from, and some cannot be sampled from at all. OT-Flow [65] iterates upon FFJORD by incorporating an optimal transport regularizer, a potential function, and other engineering optimizations which ultimately allows it to train roughly 20x faster than FFJORD with better performance; for these reasons, we use OT-Flow as our generative model.

Maximum Mean Discrepancy

Most normalizing flows use the loss C for evaluation. The loss C is used to train the forward flow and the testing set loss should provide the same qualification, however, as reported in OT-Flow [65] there are cases where testing loss is low even when the flow is poor. Furthermore, as flows are invertible this also implies the data generated by the reverse flow ρ_0 is also poor. For this reason, we use maximum mean discrepancy (MMD) [27] to evaluate the flow:

$$\mathrm{MMD}(X,Q) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} k(x_i,x_j) + \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} k(q_i,q_j) - \frac{2}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} k(x_i,q_j) \tag{3.13}$$

where $X = \{x_i\}_{i=1}^{N}$ are samples from ρ_0, and we generate $Q = \{q_i\}_{i=1}^{M}$ where $q_i = f^{-1}(y_i)$ and we use a Gaussian kernel $k(x_i, q_j) = \exp(-\frac{1}{2}||x_i - q_j||^2)$. MMD tests the difference between these two distributions (ρ_0 and our flow estimate of ρ_0). A low MMD value indicates that the two distributions, real samples and synthetic samples are likely to have been drawn from the same distribution [27]. We do not use MMD in training and only use it at the end to evaluate the effectiveness of our generative modeling efforts.

Interpreting C

The loss C mentioned above is one of three components that are optimized, the remaining two terms are regularizers whose weights α_1, α_2 require finetuning. We use a hidden space of size $m = 256$, $n_t = 8$ steps in the Runge-Kutta solver, and 2 ResNet layers for the potential model Φ for all experiments. The loss C is derived from the KL divergence between ρ_0 and ρ_1, the full derivation is available in [65]. A low C does not imply good quality generation, but it does indicate whether or not training was successful. Specifically, we seek to minimize the KL divergence between our flowed distribution and target distribution

$$\mathbb{D}_{KL} = \mathbb{E}_{\rho_0(x)}\{\log(\rho_0(x)) + C(x,T)\} \tag{3.14}$$

where the flowed distribution at time T is denoted $\rho(x,T)$ and our target distribution is the normal distribution. Specifically, we want this value to be small, so we expect a negative value for C since $\log(x_0)$ is an increasing function. This can be misleading as the only negative term in $C(x,T)$ is $l(x,T) = \log\det\nabla f(x)$. It is important to note that this gradient is not a loss surface, rather, it describes the change in density from ρ_0 to our target Gaussian distribution.

Unlike our conditional normalizing flow with image data, there is no an easy way for us to incorporate conditional information into the coupling layers as we do on CT-Scans with data features [58]. Specifically, with tabular datasets, time-series features are important, and there has been work done in incorporating past hidden state and attention into the coupling layers [79, 70], but there is no easy way for us to concatenate it into the flow since the network parameters decide the coupling patterns and how frequently they occur.

This begs whether our data will have any benefit to our downstream task. Specifically, if we are not adding any new information to our generative model, we must only add noise that will certainly stunt performance. However, we believe that OT-Flow shines through in this application because it allows us to generate from minimal data. This means the generator is learning to extract the most salient features from $\rho_0 = \pi_\beta$ such that with a synthetically augmented dataset the net signal-to-noise ratio is still very low. Thus our artificially generated data functions as feature engineering to de-noise our real data. Using a planning model leads to better estimates of the underlying transition function and, accordingly, better and faster policy convergence.

3.5 Online Reinforcement Learning

We begin with a naive approach to using PPO to learn policies using purely online data by deploying the model in the real-world microgrid environment. This approach is classic online reinforcement learning, and we refer to this as Online PPO.

3.6 Approximating Prosumer Response

Offline data does not work out of the box with on-policy training, so we explore leveraging the offline data from π_β to implement a planning model. The critical component of this planning model is an approximation of the prosumer demand response function, as described in Equation 3.6. If our approximation function is fitted to the offline data $f' \approx f$, we filter out actions that may lead to lower rewards during the policy gradient. We do this by using f' to provide an approximate distribution of prosumer response from the action distribution, and then we apply the reward function and check if it corresponds to positive net cashflow. If the reward approximation indicates the action as unprofitable, we resample a different action.

We try two approaches to learn this approximation function. The first uses 10 fully connected neural networks $\phi_1, \phi_2, \ldots, \phi_{10}$ (batch normalized, with ReLU activations, details provided in Appendix B) to learn f. The networks receive a_t, t and predict p_t. The neural network approach requires fine-tuning, so we also tried querying the offline history for a response. We queried the offline response for a matched prior experience in several different ways: no-history, full-day, and k-hour history. No-history builds up the day response by querying the offline data one hour at a time $(a^{buy}(h), a^{sell}(h))$, full-day queries for the entire 48-dimensional a_t, and k previous hours queries for $(a^{buy}(h-k), a^{sell}(h-k), \ldots, a^{buy}(h), a^{sell}(h))$. We implement KD-Trees with caching to query the offline data efficiently, but this approach fails to scale well when $\pi_{\beta^\star}$ contains more than 30 years of history, so in our experiments, we elect to use the neural network approach.

3.7 Offline-Online RL

With the online PPO procedure, we collect 25 years worth of training data, and we denote the policy-induced by this data as π_β. The aggregate offline data has 121 columns: 48 for u_t, 48 for a_t, 24 for p_t, and one to record each day. Our goal is to train the best agent using a limited amount of data to reflect the real-world setting. We should also note that although the utility prices theoretically provide bounds on the agent action space, the agent may violate these bounds and offer non-competitive prices to express the action of doing nothing and effectively sit out that day; future work could entail structuring the action space so that such actions do not happen ever. Although PPO is an on-policy algorithm, we leverage state transitions that originate from π_β in a planning model to train a new policy. We propose training PPO in the planning model for k steps and then deploying this warm-start policy in the real world. The planning model is parameterized by $\phi_1, \phi_2, \ldots, \phi_{10}$ and trained on the offline dataset of state transitions collected from our previous simulation combined with our synthetically generated simulations. We hypothesize that this will warm-start the policy network and enable the agent to learn a profitable policy with minimal real-world steps. This process decreases the data cost and learning cost induced by training the model. We refer to this procedure as Offline-Online PPO, motivated by [42], because this network is first trained fully on synthetic offline data before transitioning to online data.

The idea of pre-training an on-policy algorithm with a behavioral cloning and tuning the policy gradient after has been explored in dexterous manipulation tasks [69]. Effectively the data from π_β will train the critic with purely random exploration, but we need to ensure the critic transfers its learning without causing it to learn from scratch again. Policy gradient methods like Actor-Critic rely on the stochasticity of the action distribution to perform exploration. This approach is susceptible to poor initialization, so in addition to planning models, we propose future works to explore alternative ways of leveraging behavioral cloning for better exploration.

3.8 Dataset Aggregation

Dataset Aggregation (DAgger) [71] is an iterative algorithm that trains a policy solely dependent on the distribution of states of the original and generated dataset. We provide pseudocode for DAgger from [4] in Algorithm 1.

Algorithm 1 DAgger Algorithm

```
Initialize $\mathcal{D} \leftarrow \emptyset$
for $i = 1$ to $N$ do
    Sample T-step trajectories using $\pi_i$
    Collect $\mathcal{D}_i = \{(s_{\pi_i}, \pi_i^\star)\}$    ▷ dataset of visited states by $\pi_i$, and actions by expert $\pi_i^\star$
    Aggregate datsets $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$
    Train policy $\pi_{i+1}$ on $\mathcal{D}$
end for
return best $\pi_i$ on validation set
```

DAgger was initially used to solve the problem of coalescing the distribution between training states from an expert behavioral policy and testing states from the RL agent. We explore interleaving offline and online training using a DAgger style approach. We incorporate the offline data in our training by alternating between taking synthetic steps in the planning model and the real-world environment, and we exponentially decay the ratio of planning model steps as training continues. We hope that the RL agent learns as much about the planning model's environment dynamics and prosumer response function as possible since there is no cost for an unprofitable agent in the planning model, and sampling data is cheap. As the agent learns from the planning model, real-world steps are slowly introduced into the dataset until we have an agent that performs well in the target environment without incurring a high learning cost. Since PPO is on-policy, we cannot mix artificial experiences in a replay buffer, instead we keep track of timestep and on synthetic steps we circumvent the step where the grid calculates the prosumer response and use our approximation function instead, we refer to this as DAgger PPO and provide an implementation in Algorithm 2. Although this approach still has an upfront 30-year data cost to train the planning model, we investigate under which circumstances this cost is worthwhile.

3.9 Guardrails

Let $\phi(a_t) \in \{0, 1\}$ denote the probability of resampling the agent's action, then $1 - \phi(a_t)$ is the probability the agent posts prices a_t for the day. This idea has been explored with application to modeling demand response with RL [44], and we employ a similar approach to theirs, but instead of using a temperature α, we use a hard condition on the threshold. Our goal is to define $\phi(a_t)$ such that our agent is able to leverage information from the offline data to avoid taking actions that will incur a loss in net cashflow. If f is the true transition function, we let f' define our planning model approximation function, and then we define the guardrail

$$\phi(a_t) = \mathbb{1}[r(f'(a_t)) > \tau] \tag{3.15}$$

where τ is the minimum reward we want to employ a_t. The indicator function tells the guardrail if the offline data suggests a_t will bring a net cashflow greater than τ. This indicator function's performance is significantly impacted by the quality of the prosumer response approximator f'. We refer to this as guardrails PPO, GR-PPO, and the implementation can be found in Algorithm 3.

Chapter 4

Experiments and Results

4.1 Healthcare Experiments

Data Collection: We conduct experiments on chest CT scan data based on the COVIDx CT-1 dataset [28].The dataset consists of 45,758 images for healthy individuals, 36,856 images for individuals afflicted with common pneumonia, and 21,395 images for individuals with COVID-19.

Pre-processing: We combine the images in the Normal and Pneumonia classes into a single Non-COVID class. We use the train, validation, and test splits defined by the official annotation files. In addition to class labels, the annotations include bounding boxes for the lungs region in the whole CT scan image. We crop the images as per the bounding box and resize them to 64×64.

Hyperparameters, Network Design and Computation used: Please refer to Appendix A and Appendix B.

Testing Procedure: We performed both quantitative and qualitative testing for conditional synthetic data generation by our model. A test set is held out from the real dataset to be used for quantitative testing. We then compare the classification performance (COVID/Non-COVID) on this test set for a classifier trained on real data vs a classifier trained on the generated synthetic data. This testing procedure is illustrated in Fig. 4.2. Since the datasets are imbalanced, we report the precision, recall, and macro-F_1 score (together referred to as classification metrics) along with the accuracy. For more information on the metrics, please refer to [39]. The closeness of the classification metrics of classifiers trained on synthetic and real data indicates an efficient design of the conditional synthetic generator. To evaluate the quality of generated samples, we report the Fréchet Inception Distance (FID) [34] for the synthetic samples. For FID calculation, we use the embeddings from our classifier trained

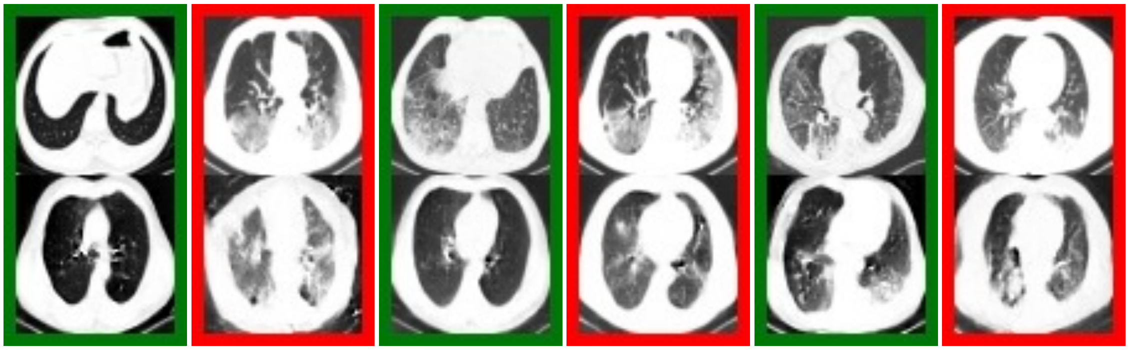

Figure 4.1: Original and generated synthetic CT scan samples. The top row consists of original samples, and the corresponding image in the bottom row is the synthetic sample obtained by preserving the original conditional feature representation and varying the local noise. Image pairs with a red border: COVID samples, and a green border: Non-COVID samples.

Model	**FID**
Ma et al.[58]	0.2504
ACGAN	0.0986
CAGlow	0.0483
Ours	**0.0077**

Table 4.1: Qualitative (Fréchet Information Distance) scores for synthetic data generated by various models (the lower the better).

using real data, in place of the official inception network [80], since the latter is not trained on medical imaging data.

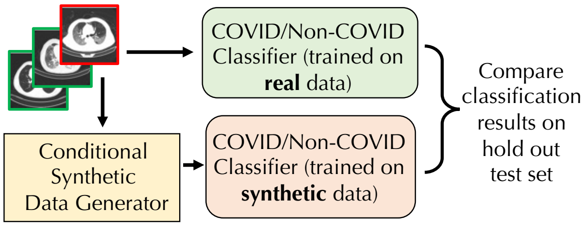
COVID/Non-COVID Classifier (trained on **real** data)
Conditional Synthetic Data Generator
COVID/Non-COVID Classifier (trained on **synthetic** data)
Compare classification results on hold out test set

Figure 4.2: Illustration of the quantitative testing procedure for conditional synthetic generation.

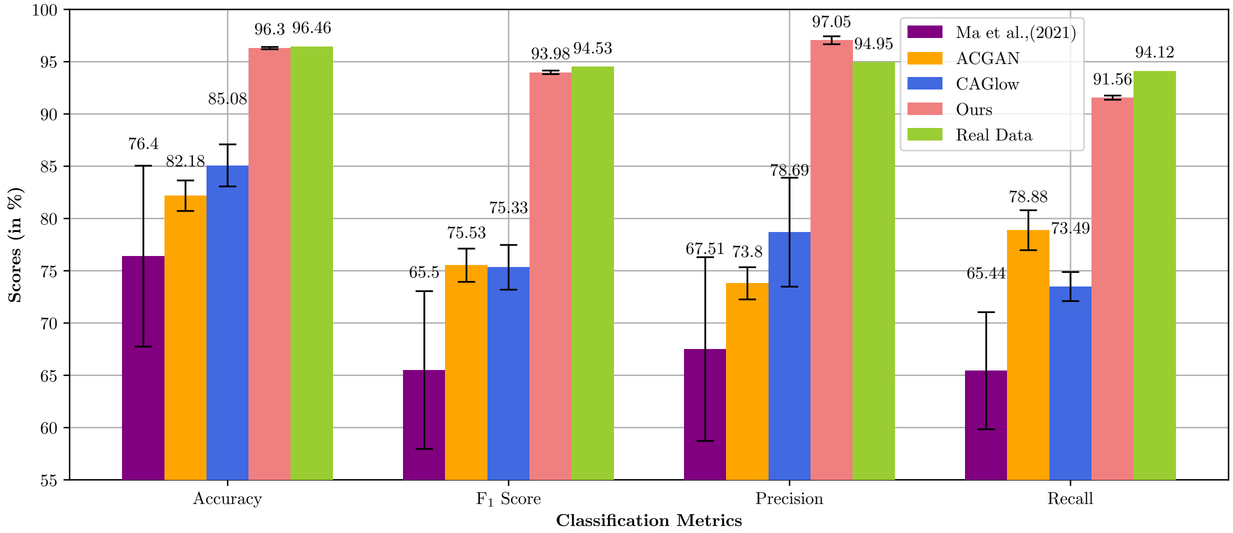

Figure 4.3: Classification metrics for classifiers trained on synthetic data generated by various models. The error bars indicate the variation in classifier performance when the synthetic datasets used to train them were generated multiple times with different seeds. A real data classifier does not involve multiple synthetic data generation, so its error bars are not included.

4.2 Results: Conditional Synthetic Data Generation

The classification results for a classifier trained on the real data vs a classifier trained on purely conditional synthetic data, and tested on a hold-out set of real data, are given in Fig. 4.3. Across the existing methods for conditional synthetic generation, the classifier trained with synthetic data from our proposed model has the closest accuracy, F_1 score, precision, and recall to that of the classifier trained on real data. This shows the capability of our method to generate synthetic samples with a distribution that closely matches the real conditional data distribution. The qualitative results (FID scores) for synthetic data generated by various models are tabulated in Table 4.1. The FID scores for our model are the lowest among all models, demonstrating that the quality of the generated samples closely matches the real ones.

It is worth noting that the accuracy/F_1 score of the classifier trained with synthetic data generated by [57] is much smaller as compared to those by other models, not to mention the classifier trained on real data. This can be justified by the fact that [57] relies on an unsupervised method of decoupling global and local information. But for conditional synthetic generation applications, such as the one presented in this paper, the model needs information on what the model designer/ domain experts consider as the conditional information (COVID/Non-COVID in our case). ACGAN and CAGlow have different generators, but both include an auxiliary supervision signal to conditionally guide the generation process. Hence, the performance of classifiers trained on synthetic data generated by them is close. We encode the conditions using feature extractors to feed to the generator, leading to state-of-the-art results.

The original samples along with the synthetic samples generated by preserving the original conditional feature representation and a different local noise for CT scans are shown in Fig. 4.1. The characteristic features for COVID CT scan samples, i.e., ground-glass opacity are well preserved in the synthetic samples. The non-conditional local features, e.g. axial plane position for CT scans are considered local noise. Since original samples for normal and pneumonia cases are merged together to form a single Non-COVID class, sometimes the corresponding synthetic image for a normal sample is a sample with pneumonia characteristics and vice-versa. This occurs since the conditional model learns to treat them as local information. As exhibited by our model, the ability to decouple the feature representations for given conditions from other information in the data should be considered the strength of an effective conditional generative model.

4.3 Results: Conditional Synthetic Generation under Label Scarcity

Previously, we proposed a semi-supervised learning approach to efficiently generate conditional synthetic samples when the number of samples labeled out of the available pandemic data is less. To test our approach, we retained the assigned label (COVID/Non-COVID) for a few samples and discarded the label for the rest of the samples. The number of labeled samples varied from 20 samples to 50 samples to 0.5%, 1%, and 5% of the total training data. The ratio between COVID and Non-COVID samples was maintained among the labeled samples. We conducted presumptive labeling and classifier training in an iterative manner and then trained the conditional generative flow using the conditional feature em-

[With different sets of labeled samples and test set bootstrapping]

Amount of labeled data	Accuracy (%)	F_1 Score (%)	Precision (%)	Recall (%)
20 samples	84.84 ± 2.91	76.32 ± 5.24	77.15 ± 4.87	76.35 ± 5.87
50 samples	90.87 ± 1.31	85.86 ± 1.73	86.48 ± 2.68	85.43 ± 1.32
0.5% of training samples	93.90 ± 0.46	90.49 ± 0.61	91.30 ± 1.28	89.8 ± 0.68
1% of training samples	95.06 ± 0.49	92.14 ± 0.69	93.94 ± 1.30	90.62 ± 0.48
5% of training samples	95.80 ± 0.20	93.24 ± 0.28	95.09 ± 0.84	91.23 ± 0.50
100% of training samples	96.30 ± 0.11	93.98 ± 0.17	97.05 ± 0.38	91.56 ± 0.20

[With multiple synthetic sets generated using random seeds]

Amount of labeled data	Accuracy (%)	F_1 Score (%)	Precision (%)	Recall (%)
20 samples	85.70 ± 0.32	78.65 ± 0.65	77.96 ± 0.49	79.48 ± 1.18
50 samples	90.74 ± 0.77	85.27 ± 0.88	86.93 ± 2.03	83.98 ± 0.68
0.5% of training samples	94.66 ± 0.86	91.41 ± 1.27	93.93 ± 2.02	89.39 ± 0.80
1% of training samples	95.04 ± 0.32	92.00 ± 0.47	94.53 ± 0.88	89.96 ± 0.42
5% of training samples	95.62 ± 0.21	92.95 ± 0.28	95.33 ± 0.77	90.99 ± 0.18
100% of training samples	96.30 ± 0.11	93.98 ± 0.17	97.05 ± 0.38	91.56 ± 0.20

Table 4.2: Results for classifiers trained on synthetic data generated by models that are developed using a few labeled data.

beddings obtained using the feature extractors. We then generated conditional synthetic data using the above trained generative model. To show the robustness of our method, we perform bootstrapping on the test set and repeat our experiments using different sets of labeled samples from the training data. For each model, we also evaluated multiple synthetic sets generated using random seeds. The results of classification models trained on the synthetic data under different bootstraps and seeds are given in Table 4.2.

As is apparent from the table, using even a few labeled samples, our method is able to achieve results on par with the case when all the labels are available. This further reinforces the strength of our approach in generating conditional synthetic data to rapidly adapt ML models to a new pandemic at its onset, when there is a scarcity of such labels. As expected, at lower levels of labeled data, the uncertainty associated with synthetic data generation is high, as is apparent from Table 4.2, which dies down as we increase the labeled data amount. The uncertainty associated with classification models trained on synthetic sets generated by our model using different seeds is low. Both the above observations establish the robustness of the proposed method.

An important point to note here is that the closeness of results obtained by utilizing 5% of labels as compared to using 100% of labels does not denounce the importance of the remaining 95% of labels. In healthcare, an improvement of even 1% of accuracy/F_1 score corresponds to a significant number of samples classified accurately, and is important, especially during a

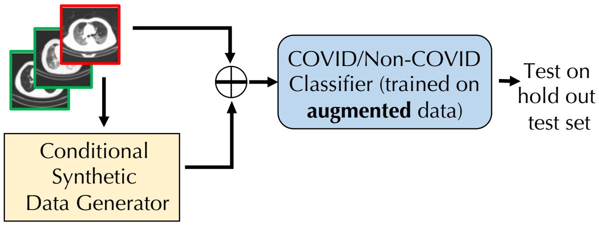
COVID/Non-COVID Classifier (trained on **augmented** data)
Test on hold out test set
Conditional Synthetic Data Generator

Figure 4.4: Illustration of synthetic data augmentation and testing process. Improvement in performance of classifiers trained on augmented data as compared to that trained on original training data is a step toward robust COVID-19 detection.

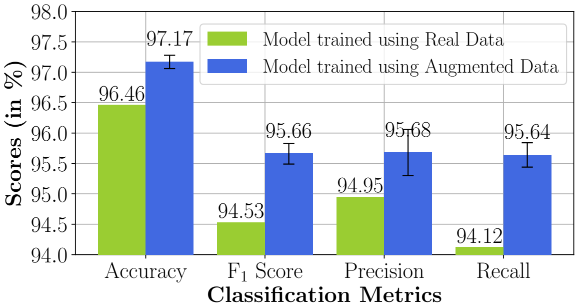

Figure 4.5: Classification results for models trained using real data (with class imbalance) vs augmented data (class-balanced). The real data (having ∼ 20% of COVID samples) was augmented with synthetically generated COVID samples using the proposed model for class balancing.

pandemic. Thus, our proposed semi-supervised approach should be considered as a remedy for cases when labels are scarce, not as an alternative to a fully-supervised approach.

4.4 Example Use of Synthetic Data: Robust Detection of COVID-19 via Data Augmentation

Generated synthetic data can be utilized in a number of downstream tasks. We conduct experiments on one of the tasks: robust detection of COVID-19 via synthetic data augmentation. The training data is inherently highly class-imbalanced, with limited samples of COVID and abundant samples for pneumonia and normal cases. To design a robust COVID-19 detection mechanism under such a class imbalance scenario, we augment the training data with synthetic COVID samples generated using the proposed model to increase the % of COVID samples and balance the dataset. The augmentation process and the testing procedure are illustrated in Fig. 4.4. The classification metrics for classifiers trained on the augmented training data are given in Fig 4.5.

Examining the classification results, the classifier trained on augmented training data has better performance than classifiers trained only on limited real training data for all augmentation levels. Note that even a slight improvement in the recall score translates to numerous samples classified correctly (e.g. 1% improvement in recall for CT scan corresponds to 200 more correctly classified samples), leading to better diagnosis and accurate and timely treatment.

4.5 Microgrid Experiments

Data-Collection: The RL agent solves an MDP defined by the state space $S :=$ (Utility prices, noisy predicted solar generation, yesterday's prosumer demand) $\in \mathbb{R}^{48+24+24}$ and action space $A := (a_{buy}, a_{sell}) =$ (energy buy prices, energy sell prices) $\in \mathbb{R}^{24+24}$. The agent's goal is to maximize a reward defined by its individual profit. We collect data by logging the training trajectory of an Online-PPO agent for 9255 days, and since we use 4 workers during training, this effectively provides us with 37,020 (a_t, p_t) rollouts of one day. We denote the policy-induced by this data as π_β.

Pre-Processing: We split the rollout data into ten separate tables. Where table j, denoted $p^{j\star}$ contains rollouts for prosumer j, specifically these are the tuples $(a_t, p^j_t, t) \in \mathbb{R}^{48+24+1}$ for each day $t \in [0, 9254]$. As illustrated in Figure 4.6, we train a normalizing flow on the real data and then generate 100,000 synthetic samples for each prosumer j by first sampling 100,000 points from $\mathcal{N}^{73}$, where $\mathcal{N}$ denotes a standard normal Gaussian and then using the inverse flow for prosumer j to generate $p^{j\star}$. Since we used real data p^j to create $p^{j\star}$, we include the real data when training the prosumer response function ϕ_j. When training the normalizing flow we normalize each feature to be between $[0, 1]$.

Hyperparameters, Network Design and Computation used: Please refer to Appendix A and Appendix B.

Testing Procedure: We test each synthetic flow's performance by calculating its MMD as defined in Equation 3.13. Once the raw synthetic data is produced we train our prosumer response predictors $\phi_1, \phi_2, \ldots, \phi_{10}$ to predict the response vector $p_t \in \mathbb{R}^{24}$ given data input $(a_t, t) \in \mathbb{R}^{49}$ using mean square error loss and validate performance on a held-out test set of 20% of the data. Once these planning model components are complete we leverage them in different ways (Offline-Online, DAgger, Guardrails) and compare weekly averaged rewards in $USD versus time across our different calibrations with a focus on both the rate and value of convergence. We run our experiments and plot the exponentially weighted average using $\alpha = 0.01$ across multiple random seeds and show standard errors.

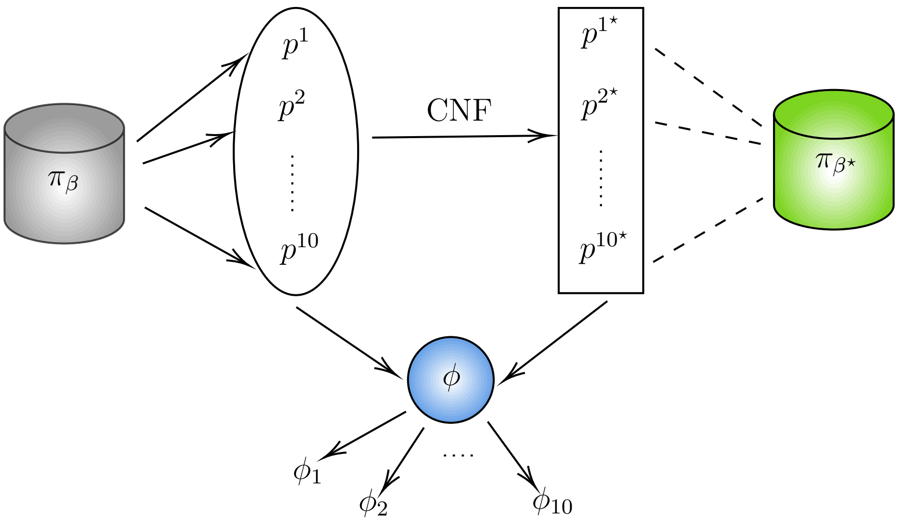

Figure 4.6: This figure illustrates our data pre-processing, specifically on the construction of prosumer response approximation networks. We begin with offline data from some policy π_β and split it up into 10 tables, one for each prosumer p^j. Then we train a continuous normalizing flow on each prosumer p^j which maps each row in the table $(a_t, p_t, t) \in \mathbb{R}^{48+24+1}$ to a 73-dimensional gaussian. Then we sample from this gaussian and use the inverse flow to generate synthetic data for each prosumer $p^{j\star}$. The 10 synthetic response tables induce a synthetic policy $\pi_{\beta^\star}$. Then we combine the real and synthetic data for each prosumer j to train the prosumer response approximation function ϕ_j.

	Benjamin	Bianca	Brian	Eileen	Elie	Benthe	Bobbi	Bryon	Elizabeth	Evelyn
MMD	0.27	0.27	0.05	0.23	0.25	0.11	0.29	0.08	0.14	0.19

Table 4.3: Flow MMD values for each prosumer as described in Equation 3.13.

We train each continuous normalizing flow using the methods described above and we report the MMD values for each prosumer flow and we report these values validating the performance of our normalizing flows in Table 4.3.

4.6 Results: Offline-Online PPO

Motivated to resolve the learning cost of a strictly online agent, we hypothesize that simply warm starting the agent by training it in a synthetic simulator for the right amount of time will prevent it from losing money once deployed in the real world. Elaborating on Section 3.7, we train the synthetic model in our planning model for the first k episodes and then we turn off our planning model for the rest of training. In the planning model our reward is

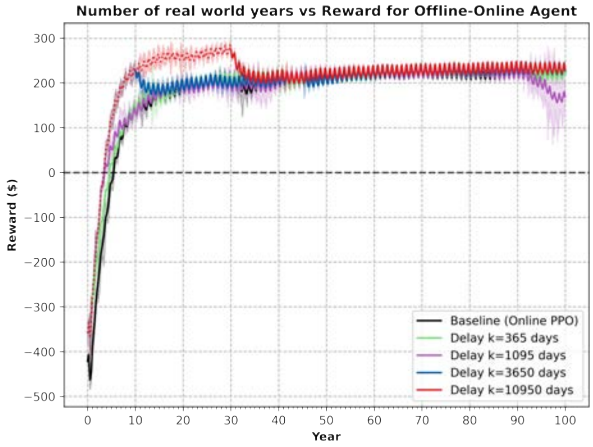

Figure 4.7: Here we look at the number of years the agent is trained versus the weekly average reward of $USD. We use the dotted line style to indicate steps in the planning model and a solid line to show steps in the real world. Using a planning model for one year and three years yields similar performance to using a planning model for ten years and 30 years. With $k \geq 3650$, there is a noticeable drop-off in reward once the agent operates strictly in the real world. We also note that the agent with $k = 1095$ has an odd drop-off in performance in its last ten years.

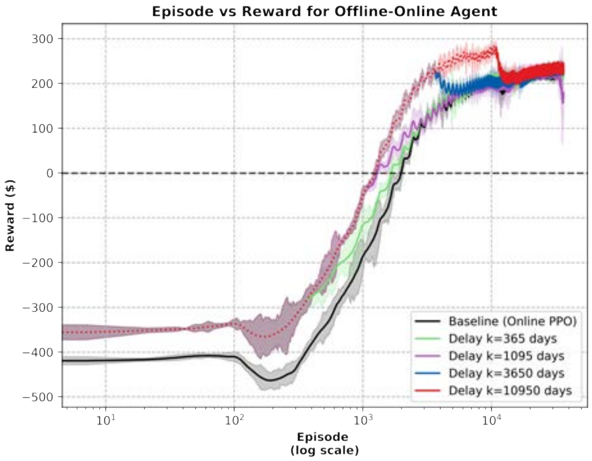

Figure 4.8: We plot Figure 4.7 with a log scale on the x-axis. We use episode as the label instead of day to not cause confusion to readers between synthetic and real days. The log scale allows us to clearly see how many days it takes for the agent to converge on a profitable policy once deployed in the real world. As we increase the amount of time the agent spends in the planning model, we observe an exponential increase in the reward that it starts with in the real world.

Figure 4.9: These figures depict the performance of different Offline-Online PPO agents against the baseline Online PPO agent. Aside from $k = 1095$, all other agents converge to the same reward once the planning model is removed. This is particularly noticeable in the drop-off at year ten and year 30 for $k = 3650$ and $k = 10950$.

parameterized by $\phi_1, \phi_2, \ldots, \phi_{10}$ so we focus on the performance of our agent after the first k episodes. It is important to take note that there is a significant oscillation in prosumer demand for a period of one year. This oscillation is mirrored in all of our agent reward curves at convergence since each prosumer's daily load and generation come from real seasonal data that parameterizes the environment.

These results in Figure 4.9 show that providing the agent with a head-start in the real world, via warm-starting, is beneficial for bypassing the learning cost the baseline agent incurs for the first 5 years. All of the Offline-Online PPO agents perform at least as well as the baseline. Each Offline-Online agent breaks a reward of 0 before the baseline and converges to the same average reward in the real world, suggesting that the planning model accurately captures the dynamics of a profitable policy. However, there are some differences between the planning model and real-world pricing dynamics as suggested by the dropoffs we observe when the planning model is removed. These dropoffs indicate that a profitable policy in the planning model is a profitable policy in the real world, but the margin of profit is actually higher in the planning model. This discrepancy highlights the distribution shift between our real offline data π_β and our synthetic offline data $\pi_\beta^\star$ and is to be expected unless our goal is to create a synthetic simulator that is a clone of the real world, which is impossible because even if we had access to the underlying transition function we would still have to capture all of the environment's stochasticity.

4.7 Results: DAgger PPO

The Offline-Online agent results are promising and validate the effectiveness of our planning model as a place for the agent to learn virtually for free. Technically, there is a financial cost of initially getting offline data for the agent. Still, it is worth it since we can train multiple profitable agents in numerous environments using offline data from one environment. These experiments investigate the drop-off we observed when turning off the planning model. We hypothesize that using a DAgger style approach will allow our agent not only to learn a profitable policy faster than the baseline (as with Offline-Online) but also to help make convergence smoother as real-world steps are gradually introduced into training. We use $\beta = 0.5$, and as illustrated in our pseudocode (Algorithm 2) we alternate between $M\beta^i$ steps in the planning model and 1 step in the real world until we are only taking real steps. For different choices of $M = [365, 1095, 3650, 10950]$ the DAgger PPO algorithm spends roughly $[2, 6, 20, 60]$ years, respectively, in the planning model.

We illustrate our results in Figure 4.12. Similar to Figure 4.8 we observe that increasing the number of steps in the synthetic model, despite the mixing scheme, leads to the model learning a profitable policy. The mixing scheme provides us with a smoother convergence to the baseline reward than the Offline-Online agent as illustrated in Figure 4.9. Although the convergence is smoother, there is no evidence that the DAgger agent learns a policy with a greater profit margin or that the agent learns a profitable pricing policy faster. This is not a negative result, but it suggests that the only advantage of using DAgger PPO over Offline-Online PPO is a smoother convergence.

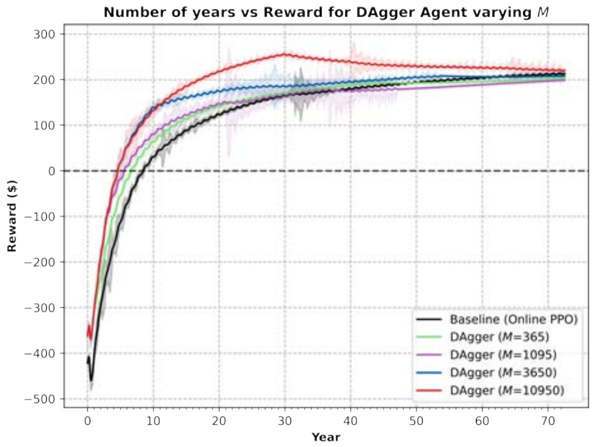

Figure 4.10: In this figure, we compare the number of years trained against the weekly average real reward in $USD.The value of M indicates the first day on which the agent takes a real step, even though it only takes one step before going back to the planning model. So we see a familiar drop-off at the 10 years and 30 years for $M = 3650$ and $M = 10950$. However, these dips are not as steep as Figure 4.7 and each of the learning curves gradually converge to the baseline reward.

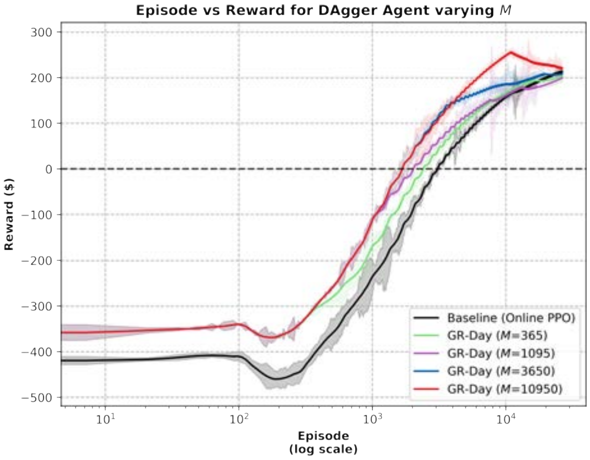

Figure 4.11: Here we look at the figure on the left with a log scale on the episodes. We are clearly able to see that increasing M results in learning a profitable policy faster. We also notice that $M = 1095$ converges to a reward value lower than the other experiments.

Figure 4.12: Unlike Figure 4.9 we do not use dotted lines to indicate synthetic steps because DAgger PPO alternates between synthetic steps and real steps until $\beta M_i < 1$. It is important to note that this means the red curve is taking steps in the planning model until year 30, the blue curve is taking steps in the model until year 10, and so on. This makes learning cost is tricky to evaluate, but we can get a rough estimate by looking at the value at decayed increments: for the red curve this would be years 30, 45, 52 and so on.

Algorithm 2 DAgger mixing procedure

```
Require: $\mathcal{D} \leftarrow \emptyset$                ▷ Initialize with empty data buffer
Require: $\pi_1 \sim \Pi$                ▷ Random initialization for initial policy
for $i = 0$ to $N$ do
    Sample one episode, $T$-timesteps using policy $\pi_i$
    for $j = 1$ to $\lfloor M_i \rfloor$ do                ▷ Taking $\lfloor M \rfloor$ synthetic steps
        $a_j \leftarrow a \sim \pi_1$
        $r'_j \leftarrow r(a, f'(a), u)$                ▷ Approximate synthetic response $f'(a) = p'$
        Collect synthetic step $\mathcal{D}_{ij} = (s_j, a_j, s_{j+1}, r'_j)$
    end for
    $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_{i0}, \mathcal{D}_{i1}, \ldots, \mathcal{D}_{i\lfloor M_i \rfloor}$
    Train $\pi_{i+1}$ on $\mathcal{D}$
    $M_{i+1} \leftarrow M_i * \beta$
end for
```

4.8 Results: Guardrails Algorithm

Our results with Offline-Online PPO and DAgger PPO validate our planning model approach illustrated in Figure 4.6 and we have successfully shown that we can use it to push the agent to learn a profitable policy faster and eliminate the real-world financial cost of learning a profitable policy. We now shift our focus to exploring additional ways to leverage the planning model for shaping the agent's learned policy. The guardrails approach explained in Section 3.9 is very similar to DAgger PPO but we parameterize the mixing procedure by threshold choice τ which is easier to interpret than our arbitrary choices of M in DAgger PPO.

After it has been generated, our planning model provides free training to the agent because there is no financial or data cost. We investigate using guardrails in two different ways: Guardrails-Cutoff and Daily-Guardrails. The former use guardrails until a cut-off after which the planning model is no longer used, the latter keeps guardrails but forces the agent to take a real step. Both of these variations use the cutoff component from Offline-Online PPO and use parameterized data mixing similar to DAgger PPO during training.

Guardrails-Cutoff

Guardrails-Cutoff (GR-Cutoff) uses guardrails until a cutoff point k after which we only take real-world steps. During the guardrails portion, our agent takes a step in the planning model, and if the planning model reward is greater than a threshold τ the agent takes this action in the real world. If the action is below a threshold the agent does not take any action for that day. After k steps in the real world, the planning model is removed. We added this cutoff component because we found that for higher values of τ the agent would never take steps

Algorithm 3 Guardrails Agent Rollout

```
Require: $\pi_\theta$    ▷ Policy with parameters $\theta$
Require: $f' : \mathcal{A}_t \rightarrow p_t$    ▷ Prosumer Response approximator
Require: $\phi : \mathcal{A}_t \rightarrow \mathbb{R} \in [0, 1]$    ▷ Guardrail function
for $i$ in $1, ..., T$ do
    Observe $s_t$
    $a_t \leftarrow a_t \sim \pi_\theta(a_t|s_t)$    ▷ Sample action from current policy
    $p \leftarrow \phi(a_t)$ ▷ Resolve guardrail on action, yielding probability this action is profitable
    if use_guardrails then    ▷ Different for GR-Cutoff and Daily-GR
        $p_t \leftarrow f(a_t)$
        $r \leftarrow r_t(a_t, p_t, u_t)$
        Rollout $(s_t, a_t, s_{t+1}, r)$
    else
        $p'_t \leftarrow f'(a_t)$
        $r' \leftarrow r_t(a_t, p'_t, u_t)$
        Rollout $(s_t, a_t, s_{t+1}, r')$
    end if
end for
```

on the same set of days which correspond to seasons of the year where prosumer generation was high. By enforcing the cutoff we will be able to observe how beneficial the guardrails are as an intelligently parameterized data mixing procedure. Because the Offline-Online agent performed better with larger k, we hypothesize that larger values of k for GR-Cutoff will result in better performance once the planning model is removed. We also believe that tuning τ is important. If the value is too small then bad actions will get approved and our model will be no different than Online PPO. It may seem that a really large τ is conservative and better since this will only rollout actions in the planning model that will yield a high reward, but the seasonal change in prosumer generation, a significant contributor to demand, may cause the agent to never learn how to act on those days where a reward of τ is unreachable. We believe that an optimal choice of τ is in the upper half of the best attainable rewards and forces the model to consistently take actions that bring a higher profit margin across all days in the year.

The rewards in Figure 4.15 are reported only on real days. This means that before the cutoff k, a collection of 365 days may be more than a year since guardrails will prevent the agent from taking real-world steps if the predicted reward is less than τ. This explains why we see higher rewards for larger values of k, the guardrails will only let the agent act when the reward is sufficiently high. This is beneficial in the beginning to close in a profitable subspace of the action space quickly, but it can be impractical because we wish to use our agent on all days, not just the ones where it will yield a larger profit. When we turn off the guardrails on the day k we are able to see that although the policy is net profitable, the

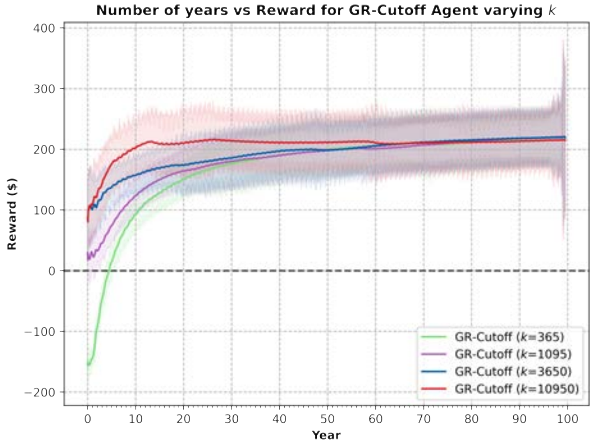

Figure 4.13: In this figure we look at the average weekly reward versus year on real days, varying k for Guardrails Cutoff agents. It is evident that increasing k corresponds to higher rewards in the beginning but after the cutoff, the rewards converge to 212 but there is a large window of standard error.

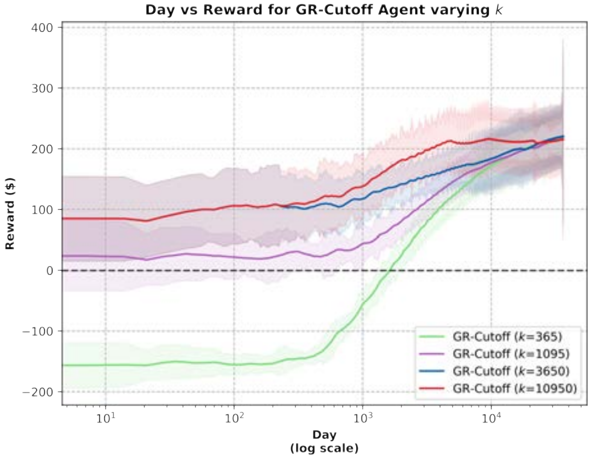

Figure 4.14: Here we look at the log-scaled day of the figure on the left. Cutoffs of $k \geq 3650$ are identical until the 10-year point at which $k = 10950$ performs better until it eventually converges back to the same value as the other curves. We also observe that the value of k has an exponentially inverse relationship with the difference between experiments in real reward. As we increase k we notice an exponentially decreasing difference in weekly average reward between the GR-Cutoff agents.

Figure 4.15: The two figures above illustrate the performance of GR-Cutoff agents aggregated across all different values of $\tau \in [-100, 0, 100, 200, 400, 500]$.

profit on the entire year smoothens out to a value similar to what the baseline achieves.

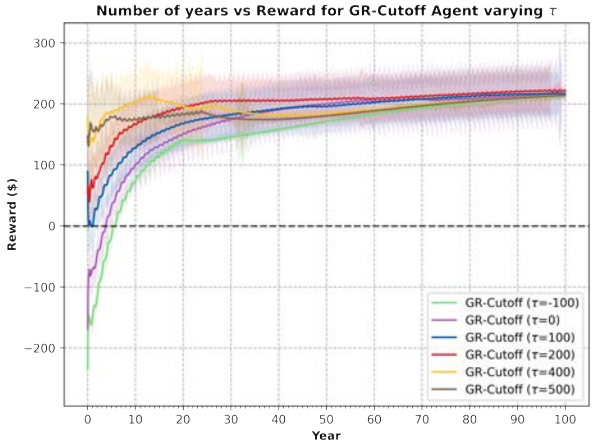

Figure 4.16: This figure shows the average weekly reward across 100 years for different values of τ. A larger value of τ leads to a larger reward, but with large values $\tau \in [400, 500]$ the net total actually performs slightly worse.

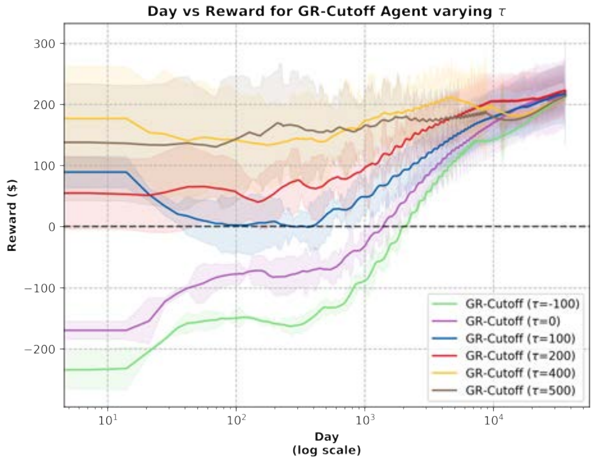

Figure 4.17: We plot the figure on the left using a log scale for the day. It is evident that larger values of τ lead to a performance improvement in the real world. Specifically, $\tau \geq 200$ with guardrails only rolls out a profitable policy, regardless of the setting of k.

Figure 4.18: The two figures above illustrate the performance of GR-Cutoff agents aggregated across all different values of $k \in [365, 1095, 3650, 10950]$.

The rewards in Figure 4.18 provide evidence that the choice of τ is very important to control how many steps it takes for our agent to become profitable in the real world. In fact, Figure 4.16 shows that if $\tau \geq 200$ then our agent will not lose any money from the start. Specifically, if our agent action receives more than \$200 in the planning model, for all of our choices of k, the agent is necessarily profitable in the real-world environment. Ultimately tuning is important for both k and τ and specific calibrations helped us gain key insights into the effectiveness of our planning model.

Daily-Guardrails

Daily-Guardrails (Daily-GR) uses an identical scheme as Guardrails-Cutoff until k real days have passed. After this point, we use the planning model every day up to 20 times. The interpretation of this is that we use Guardrails as usual to warm-start the algorithm, and after the cutoff, we require the agent beat a simulation before posting prices to the real world. After the cutoff, the agent is required to take an action every day and has up to 20 attempts after which the last action is posted to the real world regardless of reward. The agent propagates gradients on every planning model step and we believe this will push the

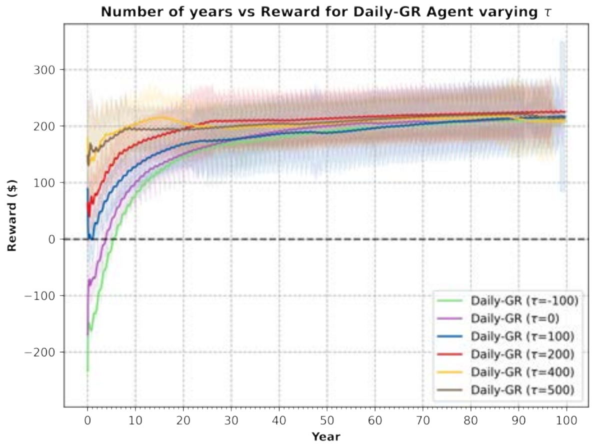

Figure 4.19: This figure illustrates the weekly average reward for each year of training using a Daily-GR agent with different values of τ. There is a direct correlation between values of τ and the reward.

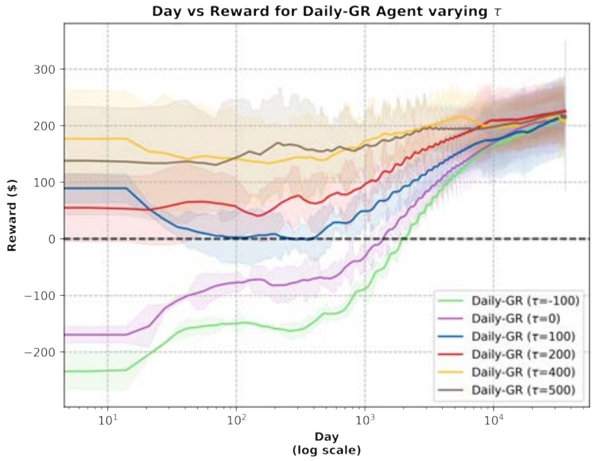

Figure 4.20: We plot the figure on the left with a log scale on the x-axis. Although all the curves converge to a similar reward value of $\tau \in [100, 200]$ seem to have marginally larger values.

Figure 4.21: These figures illustrate the performance of the Daily-GR agent with different values of τ.

agent to converge on a policy that exceeds threshold τ since we take up to 20 steps for every real step after the cutoff. If the planning model is accurate we hypothesize the agent will perform the best under this use of the planning model, since it incorporates the best components of all previous experimental variations: warm-start, intelligent data mixing, and guardrails. Even if the planning model is not completely accurate, we know that our planning model does a good job of capturing the dynamics of a net profitable policy, so at the very least we believe that the right choice of τ will ensure we perform as well as GR-Cutoff.

Overall we observe similar trends to GR-Cutoff (Figure 4.18), which makes sense since we employ the exact same training scheme until the cutoff. After the cutoff, we expect better convergence since we are taking up to 20 additional steps in the planning model for every real step and this is evident in Figure 4.20 as the curves appear to reach their converged reward more smoothly than Figure 4.17.

We would expect $k = 1095$ to be profitable and the standard error dipping below 0 in Figure 4.23 is likely due to the fact we are averaging over $\tau = -100$. Overall the trends we observed with GR-Cutoff are consistent with Daily-GR, but we observe a smoother conver-

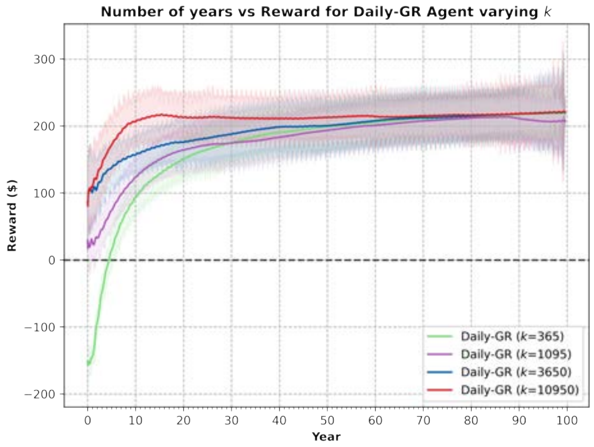

Figure 4.22: Here we have the average weekly reward versus year for the Daily-GR agent for different settings of k. This data shows that larger values of k correspond to higher rewards which eventually converge to the same value.

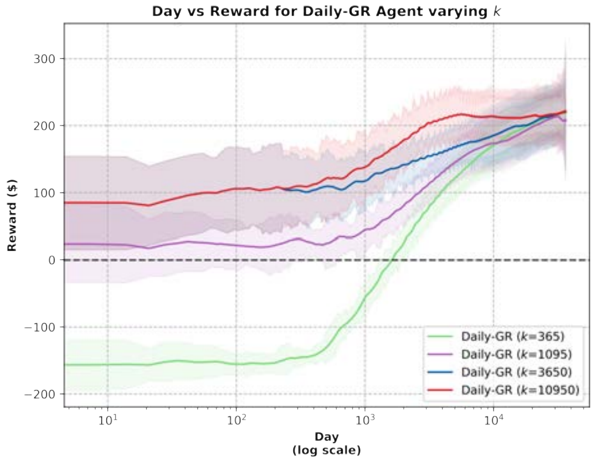

Figure 4.23: This plot takes the figure on the left and plots the x-axis on a log scale. We are able to see clearly on which day the agent's policy becomes profitable. This data shows that $k \geq 3650$ is always profitable.

Figure 4.24: We illustrate the performance of the Daily-GR agent with different values of k in the figures above.

gence since we are able to take up to take more steps between each real step.

The performance of this approach best illustrates the transferability of our planning model since our actions for Daily-GR with τ are shaped to almost certainly post actions that exceed τ in the planning model. Since higher values of $\tau \in [400, 500]$ do not translate to the best performance in the real world this can mean two things: our planning model does not perfectly capture the dynamics of prosumer response for the price settings, and that it is impossible to exceed τ on certain days. The environment settings confirm the second claim since the prosumer demand oscillates within one year depending on solar generation which itself is dependent on the season. Our results in both Figure 4.25 and 4.26 support the first claim: although the planning model is able to accurately capture the dynamics of a profitable policy, the actual threshold dynamics are best represented for $\tau \in [100, 200]$. One explanation for this is that our original offline data π_β comes from an agent trained for 30 years and during this time the agent converges to an average reward of 200. Although our planning model can extrapolate on regions of the policy it may have explored briefly, it cannot be expected to capture those dynamics it has not seen at all during the initial data collection period.

4.9 Results: Guardrails vs DAgger vs Offline vs Online for Demand Response

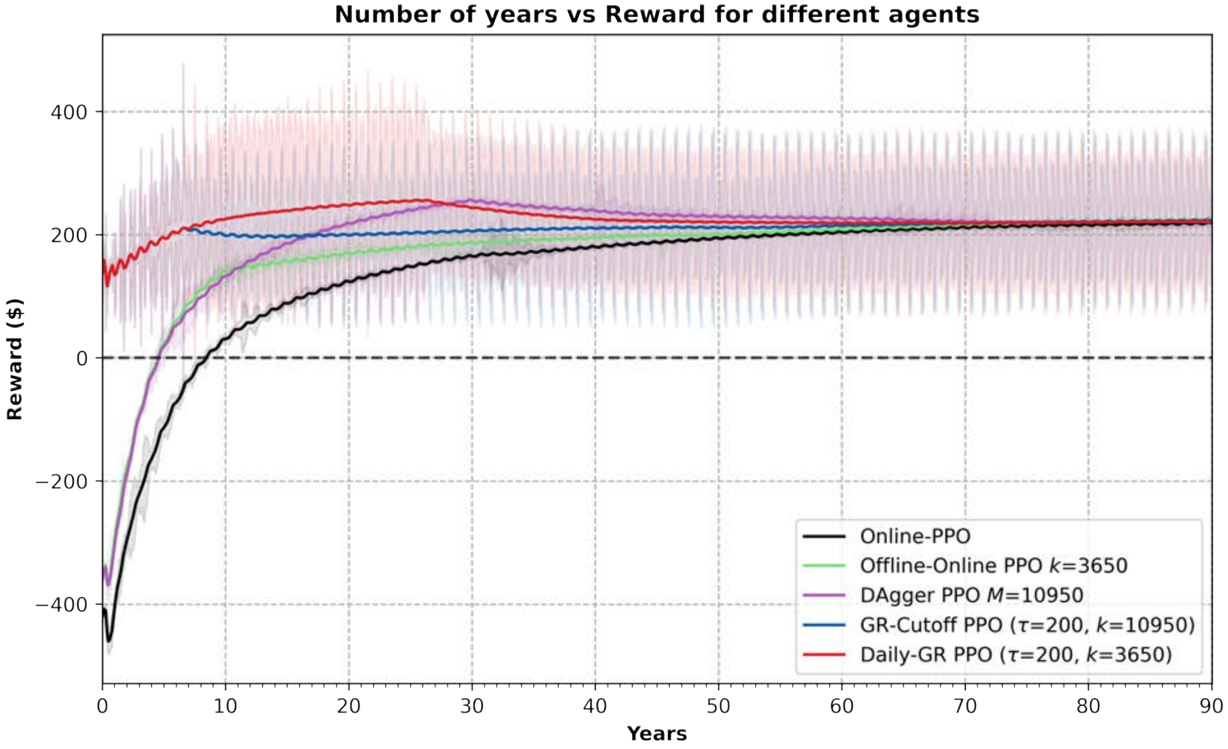

Figure 4.25: In this figure, we compare the average weekly reward versus year for the best configuration of each agent against the baseline. All the agents converge to the same average reward but Daily-GR, GR-Cutoff, and DAgger upper bound this convergence. It is important to note that although both Guardrails are showing actual real-world steps, DAgger and Offline-Online alternate between real and planning model steps so those models may have some days at the beginning where they actually incur a loss, whereas Guardrails with these choices is always profitable.

Figure 4.25 and Figure 4.27 clearly illustrate the advantage of both Guardrails approaches over the other offline approaches. Specifically, the margin of profit is larger in earlier steps and there is no cost of learning since guardrails use the planning model to decide how long the agent needs to get ready for the real world prosumer responses. We observe that both guardrails perform best with $\tau = 200$ which makes sense since our offline data π_β has taken the most steps in this range. A perfect planning model would guarantee that actions exceeding τ in the planning model also yield a reward greater than τ in the real world. For the reasons mentioned earlier this is almost impossible, but our goal is to get close. Our planning model gets within \$100 of predicting whether the reward will exceed $\tau = 200$, and

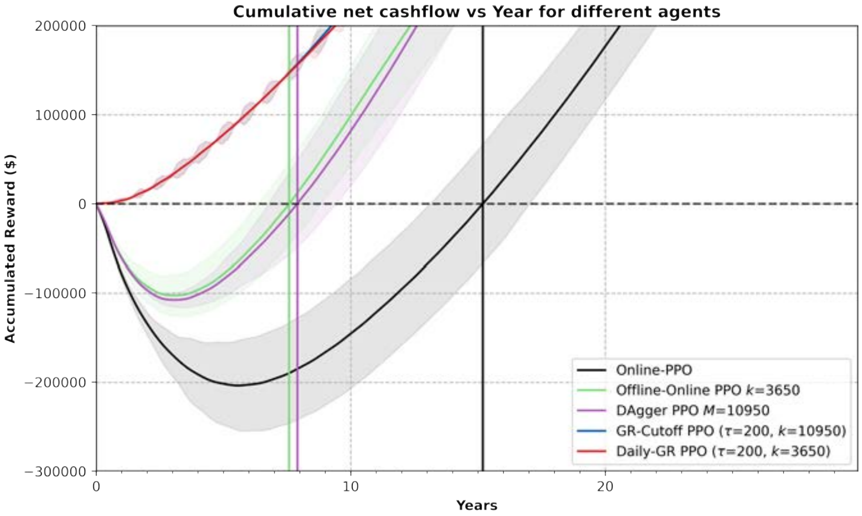

Figure 4.26: The accumulated financial liability of the agent versus years stepped for different agents. This plot illustrates the total training time each agent takes to become profitable and can be measured in real-world steps. Both Guardrails agents only report rewards on real-world steps and we observe they are always profitable in the real world. The DAgger agent takes roughly 7.9 years until it is accumulated reward is profitable, and the baseline requires 15.18 years. The Offline-Online Agent requires 7.57 years but since $k = 3650$ the first 10 years are in the planning model so the real-world accumulated financial liability is 0, equivalent to both Guardrails agents.

really close in terms of capturing profitable dynamics. Based on our plots it is evident that the planning model accurately captures the dynamics of the environment to be used to train a profitable policy (profit ≥ 0), even if the margin of profit is not completely accurate.

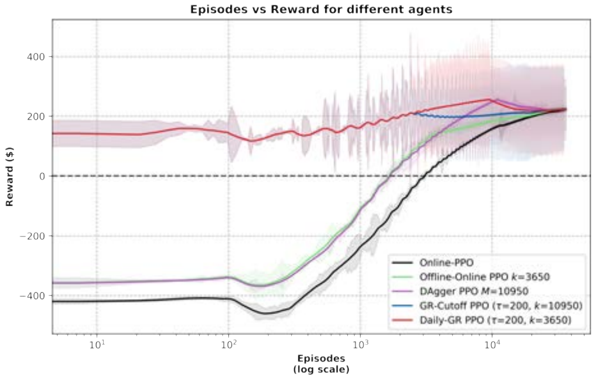

Figure 4.27: This figure shows Figure 4.25 with a log-scale on the Episode axis. Episodes are interchangeable with days but to prevent confusion with the data mixing in DAgger we list episodes.

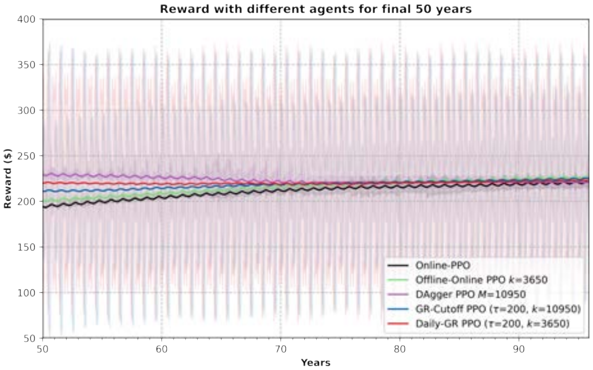

Figure 4.28: In this graph, we look at the weekly reward for the last 50 years. We notice that DAgger seems to perform marginally better, but when the steps are strictly real (after year 60) the performance converges to the rest of the models.

Chapter 5

Discussion and Future Works

We believe artificially generated samples cannot bring any new information into the dataset; rather synthetically generated data is just a noise-filtering technique that helps downstream tasks draw attention to the most essential components of the task. We hypothesize that the act of synthesizing data may be a feature engineering technique the same way re-sampling is a filtering technique. We propose future work to investigate this further. Specifically, we hypothesize that the performance of a downstream objective trained and tested solely on synthetic data compared to a real data baseline may provide better insight into the role of the synthetic data. We believe better performance in the purely synthetic task may provide evidence supporting this noise-filtering hypothesis.

Although we were able to conditionally generate CT-Scans, the actual information used to construct these samples were features embedded in the real data. Our synthetic sampling technique quite directly supports the argument that good artificial samples are just ablations of noise around various combinations of those features (both clinically visible and latent). Similarly, in the energy demand response application, the underlying signal is the stochastic process facilitating prosumer demand response. Our generation technique tries to capture that underlying signal and provide synthetic samples that maintain a high signal-to-noise ratio for a downstream data-driven task to benefit from. In the demand response setting the reinforcement learning loop contains an environment response in between the agent's action and the returned environment reward. This environment response is a distribution of variables and phenomena which the agent does not directly observe. Our approach with the flow models forces us to exemplify the relevant environment responses and repeat many runs on them via the planning model. By manipulating which runs are exemplified via conditional generation, we can control for desired behavior and influence the agent's policy search. We believe this is an exciting direction for future work along with further exploration into time-series feature extraction for offline data and ablation with different offline RL methods.

Chapter 6

Conclusion

We presented a novel conditional synthetic generative model to multiply the samples of interest at the onset of a pandemic. We conducted extensive experiments on the chest CT scan dataset to show the efficacy of the proposed model and improvements in COVID-19 detection performance achieved via synthetic data augmentation. We also proposed and experimented on a semi-supervised learning approach to efficiently generate conditional synthetic data in label scarce conditions. One of the limitations of our proposed method is that it does not exert selective control over the local noise, which can sometimes contain information for important interactions in the data, e.g., in our experiments, we extracted conditional information salient to COVID/Non-COVID, whereas the information corresponding to everything else, such as CT scan axial positions, variations of pneumonia, etc. are all considered to be the noise for the model. In general, this can be attributed to the way conditional generative models e.g. ACGAN, CAGlow function.

We also experimented with continuous normalizing flows to summarize energy price responses and constructed a planning model to simulate steps in an online RL framework. We performed extensive experiments to show the performance of our planning model and improvements in the price controller profitability. Our method takes a finite number of data points on prosumer demand, given price signals, and trains an RL agent to profitably set prices immediately without requiring online training in the environment. This approach is a unique use of synthetic data augmentation in reinforcement learning but from an information theory standpoint, it is limited by the span of the underlying signal provided in the offline data we collect.

Bibliography

[1] Daniel Adelman and Canan Uçkun. "Dynamic Electricity Pricing to Smart Homes". In: *Operations Research* 67.6 (Nov. 2019). Publisher: INFORMS, pp. 1520–1542. ISSN: 0030-364X. DOI: 10.1287/opre.2019.1882. URL: https://pubsonline.informs.org/doi/abs/10.1287/opre.2019.1882.

[2] René Aïd, Dylan Possamaï, and Nizar Touzi. "Optimal electricity demand response contracting with responsiveness incentives". In: (Oct. 22, 2018). DOI: 10.48550/arXiv.1810.09063. URL: https://arxiv.org/abs/1810.09063v3.

[3] William Arnold et al. "Adapting Surprise Minimizing Reinforcement Learning Techniques for Transactive Control". In: *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*. e-Energy '21. New York, NY, USA: Association for Computing Machinery, June 22, 2021, pp. 488–492. ISBN: 978-1-4503-8333-2. DOI: 10.1145/3447555.3466590. URL: https://doi.org/10.1145/3447555.3466590.

[4] Alexandre Attia and Sharone Dayan. *Global overview of Imitation Learning*. arXiv:1801.06503. type: article. arXiv, Jan. 19, 2018. arXiv: 1801.06503[cs,stat]. URL: http://arxiv.org/abs/1801.06503.

[5] John O. Awoyemi, Adebayo O. Adetunmbi, and Samuel A. Oluwadare. "Credit card fraud detection using machine learning techniques: A comparative analysis". In: *2017 International Conference on Computing Networking and Informatics (ICCNI)*. 2017 International Conference on Computing Networking and Informatics (ICCNI). Oct. 2017, pp. 1–9. DOI: 10.1109/ICCNI.2017.8123782.

[6] Nayana Bannur et al. "Synthetic Data Generation for Improved covid-19 Epidemic Forecasting". In: *medRxiv* (2020). DOI: 10.1101/2020.12.04.20243956. eprint: https://www.medrxiv.org/content/early/2020/12/07/2020.12.04.20243956.full.pdf. URL: https://www.medrxiv.org/content/early/2020/12/07/2020.12.04.20243956.

[7] Eder Baron-Prada, César A. Uribe, and Eduardo Mojica-Nava. "A Method for Distributed Transactive Control in Power Systems based on the Projected Consensus Algorithm". In: (Sept. 3, 2018). DOI: 10.48550/arXiv.1809.00712. URL: https://arxiv.org/abs/1809.00712v1.

[8] Justin Boyan, Dayne Freitag, and Thorsten Joachims. "A Machine Learning Architecture for Optimizing Web Search Engines". In: (), p. 8.

[9] Jason Brownlee. *A Gentle Introduction to Generative Adversarial Networks (GANs)*. Machine Learning Mastery. June 16, 2019. URL: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/.

[10] CDC. *Healthcare Workers*. Centers for Disease Control and Prevention. Feb. 11, 2020. URL: https://www.cdc.gov/coronavirus/2019-ncov/hcp/testing-overview.html.

[11] Ricky T. Q. Chen et al. *Neural Ordinary Differential Equations*. arXiv:1806.07366. type: article. arXiv, Dec. 13, 2019. arXiv: 1806.07366[cs,stat]. URL: http://arxiv.org/abs/1806.07366.

[12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[13] Emmanuel Gbenga Dada et al. "Machine learning for email spam filtering: review, approaches and open research problems". In: *Heliyon* 5.6 (June 1, 2019), e01802. ISSN: 2405-8440. DOI: 10.1016/j.heliyon.2019.e01802. URL: https://www.sciencedirect.com/science/article/pii/S2405844018353404.

[14] Hari Prasanna Das, Pieter Abbeel, and Costas J Spanos. "Dimensionality reduction flows". In: *arXiv preprint arXiv:1908.01686* (2019), pp. 1–10.

[15] Hari Prasanna Das et al. "Do Occupants in a Building exhibit patterns in Energy Consumption? Analyzing Clusters in Energy Social Games". In: *NeurIPS 2020 Workshop on Tackling Climate Change with Machine Learning*. 2020. URL: https://www.climatechange.ai/papers/neurips2020/65.

[16] Hari Prasanna Das et al. *CDCGen: Cross-Domain Conditional Generation via Normalizing Flows and Adversarial Training*. 2021. arXiv: 2108.11368 [cs.CV].

[17] Emiliano De Cristofaro. "An Overview of Privacy in Machine Learning". In: (May 18, 2020). DOI: 10.48550/arXiv.2005.08679. URL: https://arxiv.org/abs/2005.08679v1.

[18] Li Deng and Xiao Li. "Machine Learning Paradigms for Speech Recognition: An Overview". In: *IEEE Transactions on Audio, Speech, and Language Processing* 21.5 (May 2013). Conference Name: IEEE Transactions on Audio, Speech, and Language Processing, pp. 1060–1089. ISSN: 1558-7924. DOI: 10.1109/TASL.2013.2244083.

[19] Laurent Dinh, David Krueger, and Yoshua Bengio. *NICE: Non-linear Independent Components Estimation*. arXiv:1410.8516. type: article. arXiv, Apr. 10, 2015. arXiv: 1410.8516[cs]. URL: http://arxiv.org/abs/1410.8516.

[20] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. *Density estimation using Real NVP*. 2017. arXiv: 1605.08803 [cs.LG].

[21] Romuald Élie et al. "Mean–field moral hazard for optimal energy demand response management". In: *Mathematical Finance* 31.1 (Jan. 2021), pp. 399–473. ISSN: 0960-1627, 1467-9965. DOI: 10.1111/mafi.12291. URL: https://onlinelibrary.wiley.com/doi/10.1111/mafi.12291.

[22] Vincent François-Lavet et al. "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends® in Machine Learning* 11.3 (2018), pp. 219–354. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000071. URL: http://www.nowpublishers.com/article/Details/MAL-071.

[23] Mathieu Germain et al. *MADE: Masked Autoencoder for Distribution Estimation*. arXiv:1502.03509. type: article. arXiv, June 5, 2015. DOI: 10.48550/arXiv.1502.03509. arXiv: 1502.03509[cs,stat]. URL: http://arxiv.org/abs/1502.03509.

[24] Amirata Ghorbani et al. *DermGAN: Synthetic Generation of Clinical Skin Images with Pathology*. 2019. arXiv: 1911.08716 [cs.CV].

[25] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[26] Will Grathwohl et al. *FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models*. arXiv:1810.01367. type: article. arXiv, Oct. 22, 2018. DOI: 10.48550/arXiv.1810.01367. arXiv: 1810.01367[cs,stat]. URL: http://arxiv.org/abs/1810.01367.

[27] Gretton et al. "A Kernel Two-Sample Test". In: *Machine Learning Research 13* (). URL: https://www.jmlr.org/papers/volume13/gretton12a/gretton12a.pdf.

[28] Hayden Gunraj, Linda Wang, and Alexander Wong. *COVIDNet-CT: A Tailored Deep Convolutional Neural Network Design for Detection of COVID-19 Cases from Chest CT Images*. 2020. arXiv: 2009.05383 [eess.IV].

[29] Bowei Guo and Melvyn Weeks. "Dynamic tariffs, demand response, and regulation in retail electricity markets". In: *Energy Economics* 106 (Feb. 2022), p. 105774. ISSN: 01409883. DOI: 10.1016/j.eneco.2021.105774. URL: https://linkinghub.elsevier.com/retrieve/pii/S0140988321006149.

[30] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. arXiv:1801.01290. type: article. arXiv, Aug. 8, 2018. DOI: 10.48550/arXiv.1801.01290. arXiv: 1801.01290[cs,stat]. URL: http://arxiv.org/abs/1801.01290.

[31] Jessica B. Hamrick et al. *On the role of planning in model-based deep reinforcement learning*. arXiv:2011.04021. type: article. arXiv, Mar. 17, 2021. DOI: 10.48550/arXiv.2011.04021. arXiv: 2011.04021[cs]. URL: http://arxiv.org/abs/2011.04021.

[32] Tianyu Han et al. "Breaking Medical Data Sharing Boundaries by Employing Artificial Radiographs". In: *bioRxiv* (2019). DOI: 10.1101/841619. eprint: https://www.biorxiv.org/content/early/2019/11/14/841619.full.pdf. URL: https://www.biorxiv.org/content/early/2019/11/14/841619.

[33] Nikos Hatziargyriou et al. "Microgrids". In: *IEEE Power and Energy Magazine* 5.4 (July 2007). Conference Name: IEEE Power and Energy Magazine, pp. 78–94. ISSN: 1558-4216. DOI: 10.1109/MPAE.2007.376583.

[34] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: 1706.08500 [cs.LG].

[35] Jonathan Ho et al. *Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design*. 2019. arXiv: 1902.00275 [cs.LG].

[36] Tao Hong, Pierre Pinson, and Shu Fan. "Global Energy Forecasting Competition 2012". In: *International Journal of Forecasting* 30.2 (Apr. 1, 2014), pp. 357–363. ISSN: 0169-2070. DOI: 10.1016/j.ijforecast.2013.07.001. URL: https://www.sciencedirect.com/science/article/pii/S0169207013000745.

[37] Tao Hong, Jingrui Xie, and Jonathan Black. "Global energy forecasting competition 2017: Hierarchical probabilistic load forecasting". In: *International Journal of Forecasting* 35.4 (2019). Publisher: Elsevier, pp. 1389–1399. URL: https://ideas.repec.org/a/eee/intfor/v35y2019i4p1389-1399.html.

[38] Tao Hong et al. "Probabilistic energy forecasting: Global Energy Forecasting Competition 2014 and beyond". In: *International Journal of Forecasting* 32.3 (July 2016), pp. 896–913. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2016.02.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0169207016000133.

[39] Mohammad Hossin and Md Nasir Sulaiman. "A review on evaluation metrics for data classification evaluations". In: *International journal of data mining & knowledge management process* 5.2 (2015), p. 1.

[40] Chin-Wei Huang et al. *Neural Autoregressive Flows*. arXiv:1804.00779. type: article. arXiv, Apr. 2, 2018. arXiv: 1804.00779[cs,stat]. URL: http://arxiv.org/abs/1804.00779.

[41] Shruti Jadon. "COVID-19 detection from scarce chest x-ray image data using few-shot deep learning approach". In: *Medical Imaging 2021: Imaging Informatics for Healthcare, Research, and Applications*. Vol. 11601. International Society for Optics and Photonics. 2021, p. 116010X.

[42] Doseok Jang et al. "Offline-online reinforcement learning for energy pricing in office demand response: lowering energy and data costs". In: *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. BuildSys '21: The 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation. Coimbra Portugal: ACM, Nov. 17, 2021, pp. 131–139. ISBN: 978-1-4503-9114-6. DOI: 10.1145/3486611.3486668. URL: https://dl.acm.org/doi/10.1145/3486611.3486668.

[43] Doseok Jang et al. "Using Meta Reinforcement Learning to Bridge the Gap between Simulation and Experiment in Energy Demand Response". In: *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*. e-Energy '21. Virtual Event, Italy: Association for Computing Machinery, 2021, pp. 483–487. ISBN: 9781450383332. DOI: 10.1145/3447555.3466589. URL: https://doi.org/10.1145/3447555.3466589.

[44] Doseok Jang et al. *Decarbonizing Buildings Via Energy Demand Response and Deep Reinforcement Learning: The Deployment Value of Supervisory Planning and Guardrails*. SSRN Scholarly Paper 4078206. Rochester, NY: Social Science Research Network, Apr. 7, 2022. DOI: 10.2139/ssrn.4078206. URL: https://papers.ssrn.com/abstract=4078206.

[45] Yifan Jiang et al. "Covid-19 ct image synthesis with a conditional generative adversarial network". In: *IEEE Journal of Biomedical and Health Informatics* (2020).

[46] Byung Gook Kim et al. "Dynamic pricing and energy consumption scheduling with reinforcement learning". In: *IEEE Transactions on Smart Grid* 7.5 (Sept. 2016), pp. 2187–2198. ISSN: 1949-3053. DOI: 10.1109/TSG.2015.2495145. URL: http://www.scopus.com/inward/record.url?scp=84946762039&partnerID=8YFLogxK.

[47] Young-Jin Kim. "Optimal Price Based Demand Response of HVAC Systems in Multizone Office Buildings Considering Thermal Preferences of Individual Occupants Buildings". In: *IEEE Transactions on Industrial Informatics* 14.11 (Nov. 2018). Conference Name: IEEE Transactions on Industrial Informatics, pp. 5060–5073. ISSN: 1941-0050. DOI: 10.1109/TII.2018.2790429.

[48] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[49] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. 2018. arXiv: 1807.03039 [stat.ML].

[50] B Ravi Kiran et al. "Deep Reinforcement Learning for Autonomous Driving: A Survey". In: *IEEE Transactions on Intelligent Transportation Systems* (2021), pp. 1–18. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2021.3054625. URL: https://ieeexplore.ieee.org/document/9351818/.

[51] Timo Kohlberger et al. "Whole-slide image focus quality: Automatic assessment and impact on ai cancer detection". In: *Journal of pathology informatics* 10 (2019).

[52] Varun Kompella et al. *Reinforcement Learning for Optimization of COVID-19 Mitigation policies*. 2020. arXiv: 2010.10560 [cs.LG].

[53] Sergey Levine et al. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. arXiv:2005.01643. type: article. arXiv, Nov. 1, 2020. DOI: 10.48550/arXiv.2005.01643. arXiv: 2005.01643[cs,stat]. URL: http://arxiv.org/abs/2005.01643.

[54] Rui Liu et al. *Conditional Adversarial Generative Flow for Controllable Image Synthesis*. 2019. arXiv: 1904.01782 [cs.CV].

[55] Renzhi Lu, Seung Ho Hong, and Xiongfeng Zhang. "A Dynamic pricing demand response algorithm for smart grid: Reinforcement learning approach". In: *Applied Energy* 220 (C 2018). Publisher: Elsevier, pp. 220–230. URL: https://ideas.repec.org/a/eee/appene/v220y2018icp220-230.html.

[56] Kai Ma, Guoqiang Hu, and Costas J. Spanos. "A Cooperative Demand Response Scheme Using Punishment Mechanism and Application to Industrial Refrigerated Warehouses". In: (Nov. 17, 2014). URL: https://escholarship.org/uc/item/34p35679.

[57] Xuezhe Ma et al. *Decoupling Global and Local Representations from/for Image Generation*. 2020. arXiv: 2004.11820 [cs.CV].

[58] Xuezhe Ma et al. "Decoupling Global and Local Representations via Invertible Generative Flows". In: *International Conference on Learning Representations*. 2021.

[59] Kushagra Mahajan, Monika Sharma, and Lovekesh Vig. "Meta-dermdiagnosis: Few-shot skin disease identification using meta-learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 730–731.

[60] Rohit Binu Mathew et al. "Chatbot for Disease Prediction and Treatment Recommendation using Machine Learning". In: *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI). Apr. 2019, pp. 851–856. DOI: 10.1109/ICOEI.2019.8862707.

[61] *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. URL: http://yann.lecun.com/exdb/mnist/.

[62] Saeid Motiian et al. *Few-Shot Adversarial Domain Adaptation*. 2017. arXiv: 1711.02536 [cs.CV].

[63] Blake Murdoch. "Privacy and artificial intelligence: challenges for protecting health information in a new era". In: *BMC Medical Ethics* 22.1 (Sept. 15, 2021), p. 122. ISSN: 1472-6939. DOI: 10.1186/s12910-021-00687-3. URL: https://doi.org/10.1186/s12910-021-00687-3.

[64] Junier B. Oliva et al. *Transformation Autoregressive Networks*. arXiv:1801.09819. type: article. arXiv, Oct. 23, 2018. DOI: 10.48550/arXiv.1801.09819. arXiv: 1801.09819[stat]. URL: http://arxiv.org/abs/1801.09819.

[65] Derek Onken et al. "OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport". In: (May 29, 2020). DOI: 10.48550/arXiv.2006.00104. URL: https://arxiv.org/abs/2006.00104v5.

[66] George Papamakarios, Theo Pavlakou, and Iain Murray. *Masked Autoregressive Flow for Density Estimation*. arXiv:1705.07057. type: article. arXiv, June 14, 2018. DOI: 10.48550/arXiv.1705.07057. arXiv: 1705.07057[cs,stat]. URL: http://arxiv.org/abs/1705.07057.

[67] *Proximal Policy Optimization — Spinning Up documentation*. URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html#id3.

[68] Deepta Rajan et al. "Self-training with improved regularization for sample-efficient chest x-ray classification". In: *Medical Imaging 2021: Computer-Aided Diagnosis*. Vol. 11597. International Society for Optics and Photonics. 2021, 115971S.

[69] Aravind Rajeswaran et al. *Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations*. arXiv:1709.10087. type: article. arXiv, June 26, 2018. DOI: 10.48550/arXiv.1709.10087. arXiv: 1709.10087[cs]. URL: http://arxiv.org/abs/1709.10087.

[70] Kashif Rasul et al. *Multivariate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows*. arXiv:2002.06103. type: article. arXiv, Jan. 14, 2021. DOI: 10.48550/arXiv.2002.06103. arXiv: 2002.06103[cs,stat]. URL: http://arxiv.org/abs/2002.06103.

[71] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*. arXiv:1011.0686. type: article. arXiv, Mar. 16, 2011. DOI: 10.48550/arXiv.1011.0686. arXiv: 1011.0686[cs,stat]. URL: http://arxiv.org/abs/1011.0686.

[72] Lars Ruthotto and Eldad Haber. *An Introduction to Deep Generative Modeling*. type: article. arXiv, Apr. 11, 2021. arXiv: 2103.05180[cs]. URL: http://arxiv.org/abs/2103.05180.

[73] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[74] Jian Shen et al. "Wasserstein distance guided representation learning for domain adaptation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 2018.

[75] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (July 6, 2019), p. 60. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[76] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: (), p. 19.

[77] Lucas Spangher et al. "Engineering vs. Ambient type visualizations: Quantifying effects of different data visualizations on energy consumption". In: *Proceedings of the 1st ACM International Workshop on Urban Building Energy Sensing, Controls, Big Data Analysis, and Visualization*. BuildSys '19: The 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation. New York NY USA: ACM, Nov. 13, 2019, pp. 14–22. ISBN: 978-1-4503-7014-1. DOI: 10.1145/3363459.3363527. URL: https://dl.acm.org/doi/10.1145/3363459.3363527.

[78] Michael R. Starke et al. *A Dynamic Simulation Tool for Estimating Demand Response Potential from Residential Loads*. Oak Ridge National Lab. (ORNL), Oak Ridge, TN (United States), Jan. 1, 2015. URL: https://www.osti.gov/biblio/1265252.

[79] Rhea Sanjay Sukthanker et al. *Generative Flows with Invertible Attentions*. type: article. arXiv, Mar. 31, 2022. arXiv: 2106.03959[cs]. URL: http://arxiv.org/abs/2106.03959.

[80] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[81] Takeshi Teshima, Issei Sato, and Masashi Sugiyama. "Few-shot domain adaptation by causal mechanism transfer". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9458–9469.

[82] Hugo Touvron et al. "Going deeper with Image Transformers". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021 IEEE/CVF International Conference on Computer Vision (ICCV). Montreal, QC, Canada: IEEE, Oct. 2021, pp. 32–42. ISBN: 978-1-66542-812-5. DOI: 10.1109/ICCV48922.2021.00010. URL: https://ieeexplore.ieee.org/document/9710634/.

[83] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[84] A. Waheed et al. "CovidGAN: Data Augmentation Using Auxiliary Classifier GAN for Improved Covid-19 Detection". In: *IEEE Access* 8 (2020), pp. 91916–91923. DOI: 10.1109/ACCESS.2020.2994762.

[85] Linda Wang, Zhong Qiu Lin, and Alexander Wong. "COVID-Net: a tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images". In: *Scientific Reports* 10.1 (Nov. 2020), p. 19549. ISSN: 2045-2322. DOI: 10.1038/s41598-020-76550-z. URL: https://doi.org/10.1038/s41598-020-76550-z.

[86] Lilian Weng. *Flow-based Deep Generative Models*. Section: posts. Oct. 13, 2018. URL: https://lilianweng.github.io/posts/2018-10-13-flow-models/.

[87] An Zhao et al. "Domain-adaptive few-shot learning". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 1390–1399.

[88] Yuxun Zhou and Costas J Spanos. “Causal meets submodular: Subset selection with directed information”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Citeseer. 2016, pp. 2657–2665.

[89] Han Zou et al. “Consensus Adversarial Domain Adaptation”. In: *AAAI Conference on Artificial Intelligence 2019*. 2019.

Appendix A

Hyperparameters

Classifier

- Batch size: 64
- Optimizer: AdamW optimizer
- Learning rate: $1e-5$
- Learning rate decay parameters: 0.99, 0.998, 0.999, 0.999, 0.9998, 0.9998 for classifiers trained on 100% of the training set, 5%, 1%, 0.5%, 50 samples, and 20 samples respectively. The decay parameter was set to 0.99 during epochs with presumptive labels during semi-supervised training.
- Weight decay rate: $1e-7$
- Beta parameters: $(0.9, 0.999)$

Conditional Generative Flow

- Batch size: 320 across 4 GPUs
- Optimizer: AdamW
- Learning rate: $5e-4$
- Learning rate decay: It had a warm-up period of 10 epochs and was decayed on an exponential schedule with decay parameter 0.99.
- Weight decay: $1e-6$

- Beta parameters: $(0.5, 0.999)$
- Temperature for Gaussian Noise Sampling: 0.9

PPO Agent

- Batch size: 28
- SGD minibatch size: 4
- Learning rate: $526e-6$
- PPO-Clip: 0.60725
- PPO-SGD-iter: 6

OT-Flow

- Batch size: 64
- Hidden dim: 256
- α_C: 100
- α_R: 15
- Learning rate: $4e-3$
- Weight decay rate: 0.08

ϕ-networks

- Batch size: 256
- Learning rate: $5e-4$
- Optimizer: Adam

Appendix B

Network Architecture

Classifier

Our classifier network is based on COVIDNet, by [85]. It is composed of lightweight projection-expansion-projection-extension (PEPX) modules. The PEPX modules consist of 1×1 convolutions for first stage projection that projects input features to a lower dimension, 1×1 to expand the features to a higher dimension different than that of the input features, a depth-wise representation of features to learn spatial characteristics with 3×3 convolutions, 1×1 convolutions to project features back to a lower dimension and finally 1×1 convolutions to extend the channel dimensionality to produce the final features. We take the dimension of the conditional input (z) to be 32, and perform l_2-normalization on it before feeding it to the conditional generative flow.

Conditional Generative Flow

We use a variant of a Glow [49] model that features a reorganized flow step, designed to reduce the number of invertible 1×1 convolutions, together with a fine-grained multi-scale architecture. Each coupling layer consists of a 3×3 convolution with ELU [12] non-linearity, a 1×1 convolution, a channel-wise summation with a condition vector, a non-linearity, and a final 3×3 convolution. The condition vector is obtained by taking the embedding of the image at the penultimate layer of our classifier and projecting it to the hidden dimension of the 1×1 convolution layer.

We use a 4-level flow, with a granularity factor $M = 4$. The first and last levels consist of 8 flow steps, and the two internal levels each consist of a sequence of 3 blocks of 8 flow steps. The hidden dimension of the affine coupling layer at each level is $24, 512, 512, 512$ in that order.

ϕ-networks

use a feed-forward network with 3 hidden layers, batch normalized, with ReLU activations. The input data is $[0, 1]$ normalized, 49-dimensional encodings (a_{buy}, a_{sell}, day) and the hidden layers have $64, 49, 24$ units and we take the mean square error loss between our prediction and the actual prosumer response.