MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Time-Series Generative Networks for Synthesizing Realistic Scenarios in Occupant-Centric Building Simulation

Chakrabarty, Ankush; Vanfretti, Luigi; Wang, Ye; Mineyuki, Takuma; Zhan, Sicheng; Tang, Wei-Ting; Paulson, Joel A.; Deshpande, Vedang M.; Bortoff, Scott A.; Laughman, Christopher R.

TR2025-043 April 02, 2025

Abstract

Occupant-centric building simulation models rely on two key factors: our understanding of the underlying physics that govern thermal dynamics, and realistic modeling of occupancy patterns and energy use within the zone of interest. While current physics-oriented building simulation models predict thermal dynamics accurately, a systematic and scalable way to generate occupancy and energy use patterns remains an open challenge despite the large amount of data collected from building sensors across academic and industry efforts. In this paper, we leverage deep generative net- works capable of learning from real building data for generating realistic occupant-centric scenarios to inform building simulations. Our ultimate goal is to assess building performance over a wide range of generated scenarios, which is currently done either by taking a small set of 'nominal scenarios' or by handcrafting specific scenarios, both of which restrict the quality of building performance assessment to a few biased use-cases. For the purpose of generating scenarios automatically, we employ a recently proposed architecture called RAFT-VG (regularized adversarially finetuned VAE-GAN) that combines the benefits of variational autoencoders (VAEs) and generative adversarial networks (GANs), and demonstrate its capacity for synthesizing a variety of signals including occupancy patterns, internal heat loads, and ambient conditions. A key feature of this neural architecture is that the generative process depends solely on a conditional decoder network. Distilling the deep RAFT-VG model to a simpler decoder for inference allows us to propose a general framework for integrating the generative model directly in Modelica. The closed-loop building performance with various generated scenarios, along with the Modelica integration, is demonstrated via simulation use-cases using the Modelica BESTEST repository.

Building Simulation 2025

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Time-Series Generative Networks for Synthesizing Realistic Scenarios in Occupant-Centric Building Simulation

Ankush Chakrabarty[a,*], Luigi Vanfretti[b], Ye Wang[a], Takuma Mineyuki[d], Sicheng Zhan[c], Wei-Ting Tang[e], Joel A. Paulson[e], Vedang M. Deshpande[a], Scott A. Bortoff[a], Christopher R. Laughman[a]

[a]*Mitsubishi Electric Research Laboratories, Cambridge, MA, USA*
[b]*Rensselaer Polytechnic Institute, Troy, NY, USA*
[c]*Massachusetts Institute of Technology, Cambridge, MA, USA*
[d]*Mitsubishi Electric Corporation Information Technology R&D Center, Ofuna, Kanagawa, Japan.*
[e]*Ohio State University, Columbus, OH, USA*

Abstract

Occupant-centric building simulation models rely on two key factors: our understanding of the underlying physics that govern thermal dynamics, and realistic modeling of occupancy patterns and energy use within the zone of interest. While current physics-oriented building simulation models predict thermal dynamics accurately, a systematic and scalable way to generate occupancy and energy use patterns remains an open challenge despite the large amount of data collected from building sensors across academic and industry efforts. In this paper, we leverage deep generative networks capable of learning from real building data for generating realistic occupant-centric scenarios to inform building simulations. Our ultimate goal is to assess building performance over a wide range of generated scenarios, which is currently done either by taking a small set of 'nominal scenarios' or by handcrafting specific scenarios, both of which restrict the quality of building performance assessment to a few biased use-cases. For the purpose of generating scenarios automatically, we employ a recently proposed architecture called RAFT-VG (regularized adversarially fine-tuned VAE-GAN) that combines the benefits of variational autoencoders (VAEs) and generative adversarial networks (GANs), and demonstrate its capacity for synthesizing a variety of signals including occupancy patterns, internal heat loads, and ambient conditions. A key feature of this neural architecture is that the generative process depends solely on a conditional decoder network. Distilling the deep RAFT-VG model to a simpler decoder for inference allows us to propose a general framework for integrating the generative model directly in Modelica. The closed-loop building performance with various generated scenarios, along with the Modelica integration, is demonstrated via simulation use-cases using the Modelica BESTEST repository.

Keywords: Generative AI, Building controls, Digital twins, Load schedules, Performance monitoring

1. Introduction

Building energy modeling (BEM) is a key technology for improving building performance through optimal design and operation [1]. For example, it is typically used to estimate peak heating and cooling loads and specify the HVAC equipment, as oversizing or undersizing equipment can lead to sub-optimal performance during operation. This is especially true with climate change, where standard design conditions may not be representative of projections for climate conditions over the next few decades. Scenario-based performance analysis should thus be conducted so that buildings can be designed to be more robust in future operational conditions, while minimizing energy consumption and ensuring occupant comfort [2].

Existing building simulation tools, such as EnergyPlus [3], have made significant contributions to the evaluation of building energy performance during design. The basic physics of heat transfer by conduction and radiation, and mass transport by advection, are well known and transcribed accurately in these modeling methods. These methods do not

*Corresponding author. Email: achakrabarty@ieee.org

focus on describing the dynamic behavior of building systems under automatic control, however, as the equipment models in EnergyPlus are largely based on (quasi) steady-state characterizations of their performance. More recently, modeling platforms such as the Modelica Buildings Library [4], have been developed based on an acausal, equation-based paradigm, which allows for simulation of the coupled dynamic behavior of building constructions and HVAC systems, including their control.

A critical challenge to further improve building energy efficiency is the energy performance gap, which refers to the deviation between the predicted building energy consumption as assessed during the design stage and the actual building energy consumption as measured in the operation stage [5]. Actual building operations are subject to many disturbances that should be characterized to better estimate the building loads [6]. As a major source of uncertainty, building occupants also produce and absorb latent, sensible, and radiative heat, and their behavior can strongly affect the performance of an HVAC system as measured by energy consumption, human comfort, indoor air quality, etc. Typical building simulation platforms must *assume* this behavior a priori. For example, many models assume a fixed or nominal occupancy level (numbers of people) and their activity schedule, represented as an input disturbance, in order to execute a simulation.

Prior work has recognized that deterministic approaches can lead to discrepancies in the simulation results [7], and efforts have been made to characterize the stochastic nature of internal disturbances. For example, [8] designed an agent-based model to create the sub-hourly occupant schedules for different types of buildings, and [9] showed that GANs can more precisely model occupant variability. However, existing methods cannot capture the heterogeneity of human behavior and other disturbances across buildings. In addition, these models run independently from the simulation engines, making it cumbersome to integrate the generated signal in the energy models and the downstream applications.

Modern advances in machine learning and deep neural networks, combined with available data collected in buildings, are quite capable of learning occupancy schedules as a statistical model. Such models can be used in a variety of building simulation scenarios, such as in simulating nominal or corner-case occupancy patterns for a particular building to identify opportunities for improvement or to construct occupancy models that can be used in building design or HVAC product development.

To address the challenge of creating realistic occupancy scenarios for BEM, we have developed a novel method that uses generative deep learning to characterize stochastic human behavior. This method is well-suited to inclusion in Modelica-based models because the core of the generative capability is in the simple decoder sub-module, which is just a multi-layer perceptron that can seamlessly inject the generated signals into the energy models without requiring the use of extra dependencies or complex mathematical operations. The proposed method thus has good performance with respect to both accuracy and implementation. In this paper, we introduce the technical details of the proposed method and demonstrate its efficacy for building performance assessment. We highlight two novel use cases that these models enable: monitoring HVAC performance and identifying performance-limiting scenarios. Such analyses, which occur early in the design process, enable time-efficient building design practices and improved building performance, whereas previous methods have not enabled the analysis of the impact of stochastic variations.

2. Problem Statement and Proposed Solution

Sophisticated simulators such as digital twins of building energy systems comprise dynamic simulation models that can be abstracted by

$$y_t = \mathcal{M}(x_t, u_t, d_t^{\mathsf{ext}}, d_t^{\mathsf{int}}), \tag{1}$$

where $x_t \in \mathbb{R}^{n_x}$ denotes a state of the simulated dynamics, $u_t \in \mathbb{R}^{n_u}$ denotes the control variables to the system such as heating, ventilation, and cooling (HVAC) or renewable energy sources, and d_t^{ext}, d_t^{int} denote disturbance inputs to the system acting either externally to the system, or internally within the system, respectively. Examples of external disturbances include weather, whereas internal disturbances are often occupant-generated, e.g., internal heat loads due to appliance use or illumination. The simulation model $\mathcal{M}$ is often based on first principles (i.e., physics-based) that accurately reflect the thermal dynamics of the building zones, but could also contain approximators such as neural networks; for exposition, we treat $\mathcal{M}$ as a black-box.

Some outputs $y_t \in \mathbb{R}^{n_y}$ of the simulation model are available for online measurement: these outputs can be used to assess the (scalar) performance output J of the closed-loop building simulation, usually by evaluating a performance function f. That is,

$$J = f(\{y_t\}_{t=0}^{T_f}), \tag{2}$$

where T_f is the total time of closed-loop simulation over which the performance output is computed. Often, these closed-loop simulations are driven by control algorithms that have been designed for nominal scenarios, e.g., by selecting d_t^{int} and d_t^{ext} to be some nominal values based on domain experience or a simplifying statistic (such as the mean) of observed data. Even if such a controller based on a nominal scenario is effectively designed, it may not perform equally well under a wide range of scenarios. For example, if the controller was designed assuming that a zone has some nominal occupancy, it may not perform adequately if the zone's occupancy is completely empty or filled to maximum capacity.

Conventional methods to construct scenarios with the specific objective of robustly evaluating the performance of HVAC system control in a specific building require significant computational effort or intuition honed by significant domain expertise. Neither of these strategies scale well if the process of selecting candidate scenarios involves the design of a large number of time-series signals and cannot be systematically deployed across different building simulation models. Our **objective** in this paper is to provide a systematic framework for generating scenarios based on relevant building data, and to drive a high-fidelity simulator to evaluate the performance of HVAC system control. For any such method to be practical, we require it to be generalizable to any simulation model, and therefore, agnostic to the type of simulation model, as long as it can be represented by (1).

To this end, we propose a novel deep generative model to learn the distribution of internal disturbances in building operations and embed the model in a high-fidelity simulation engine, i.e., Modelica, for control evaluation. Assuming we have available a simulation model $\mathcal{M}$ and access to real scenario data $\mathcal{D} := \{d_t^{\text{int}}, d_t^{\text{ext}}\}$, we propose the use of deep generative models to learn the underlying distribution of this data. In this work, we consider a combination of a variational autoencoder (VAE) fine-tuned by a discriminator as in GANs; we refer to our architecture as RAFT-VG [10] and will describe this generative network in more detail in the next section. The RAFT-VG is trained to learn a distribution in its latent space that, when decoded, is 'close' (in some metric) to the true data distribution. Additionally, the generative model allows conditioning, that is, the distributions that are learned can be conditioned on relevant factors (e.g., workday/holiday, seasonality) that are expected to change the underlying data distribution. Therefore, sampling from this latent space and passing the latent sample through a trained decoder network results in synthetic d_t^{int} scenarios, such as internal heat loads and occupants in building zones, as well as d_t^{ext}, such as solar irradiance patterns and ambient conditions, that can drive closed-loop building simulations.

3. Generative Modeling

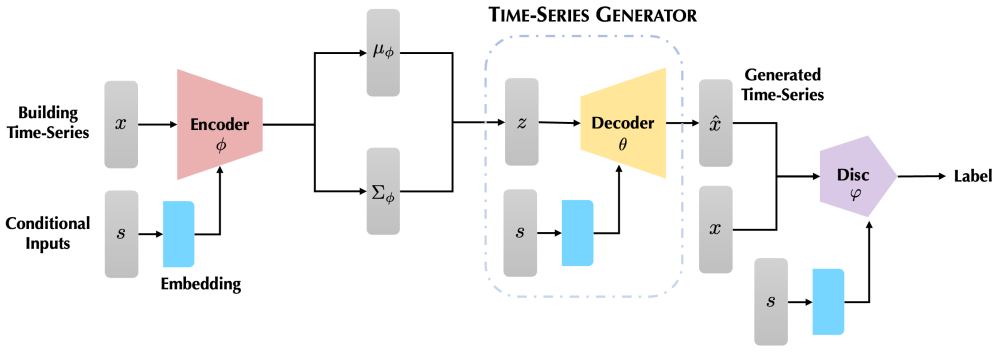

Figure 1: Proposed RAFT-VG generative network. In phase 1, the encoder-decoder pair is trained like a standard conditional VAE with the discriminator frozen. In phase 2, the encoder is frozen, and the decoder-discriminator pair is fine-tuned adversarially like a GAN.

In this work, we adopt the generative modeling approach of [10], which applies regularized adversarial fine-tuning

(RAFT) that combines the benefits of Variational AutoEncoders (VAE) [11, 12] and Generative Adversarial Networks (GAN) [13, 14]. This method is a two-stage approach, where we first train a conditional VAE in a conventional manner, and then we fine-tune the decoder with adversarial training against a discriminator, in a manner similar to Wasserstein GAN (WGAN) [15] with spectral normalization [16]. Additionally, the second-stage fine-tuning is regularized by limiting how much the decoder parameters may change, which aims to preserve consistency with the encoder. This approach combines the relative ease of training a VAE in the first stage, with the generative quality benefits of GAN-like adversarial fine-tuning in the second stage, while the regularization helps to preserve VAE consistency and improve the stability of fine-tuning. In the following, we briefly describe the steps of this method, while further details can be found in [10].

The first stage consists of training a conditional VAE that implicitly captures the conditional distribution $p_\theta(w|s)$, where $w := \{d_t^{\mathsf{int}}, d_t^{\mathsf{ext}}\}_{t=0}^{T}$ is shorthand for the disturbance time series being modeled and s is a conditioning variable related to that time series. The conditional VAE provides a generative model specified by the distribution $p_\theta(w|s,z)$, where z is the latent representation sampled from a latent prior distribution $p(z)$. Together, these distributions implicitly (and intractably) determine the conditional model distribution,

$$p_\theta(w|s) = \int p_\theta(w|s,z)p(z)\,\mathrm{d}z.$$

In principle, training the VAE aims to maximize the expected log-likelihood, i.e., $\max_\theta \mathsf{E}[\log p_\theta(w|s)]$, where the expectation is with respect to distribution of the data samples (w,s). However, since $p_\theta(w|s)$ is intractable to evaluate, the actual training object is to maximize a variational lower bound of this expected log-likelihood, also known as the evidence lower bound (ELBO), given by

$$\mathsf{E}[\log p_\theta(w|s)] \geq \mathsf{E}\left[\log p_\theta(w|s,z) + \mathsf{KL}\big(q_\phi(z|w,s)\|p(z)\big)\right],$$

where $q_\phi(z|w,s)$ is introduced as a variational approximation of the (intractable) posterior

$$p_\theta(z|w,s) = p(z)p_\theta(w|s,z)/p_\theta(w|s)$$

and the right-hand side is maximized with respect to both the parameters θ and ϕ.

The variational posterior $q_\phi(z|w,s)$ can be viewed as a probabilistic encoder that maps from the data features w to the latent representation z, while conditioned on s. We adopt the typical approach of parameterizing the encoder as a conditional Gaussian, $q_\phi(z|w,s) = \mathcal{N}(z;\mu_\phi(w,s),\Sigma_\phi(w,s))$, where the mean vector μ_ϕ and diagonal covariance matrix Σ_ϕ are parametric functions (i.e., realized by a neural network) of (w,s). With the latent prior set to the standard Gaussian distribution, i.e., $p(z) = \mathcal{N}(0,I)$, the Kullback-Liebler (KL) divergence term of the ELBO objective is tractable and differentiable [11]. Similarly, we also realize the decoder as a conditional Gaussian, i.e., $p_\theta(w|s,z) = \mathcal{N}\big(w;\hat{w}_\theta(z,s),\Sigma\big)$, where Σ is a trainable diagonal matrix, and $\hat{w}_\theta(z,s)$ is the decoder output. Thus, the first term of the ELBO is given by

$$\mathsf{E}[\log p_\theta(w|s,z)] = \mathsf{E}\left[c - \frac{1}{2}\log(\mathsf{det}(\Sigma)) - \frac{1}{2}(w-\hat{w}_\theta(z,s))^\top\Sigma^{-1}(w-\hat{w}_\theta(z,s))\right],$$

where $c = -\frac{d}{2}\log(2\pi)$ (with d as the dimensionality of w) is a constant that does not affect the optimization. The training objective for the VAE then becomes

$$\min_{\theta,\phi,\Sigma}\ \mathsf{E}\left[\frac{1}{2}\log(\mathsf{det}(\Sigma)) + \frac{1}{2}(w-\hat{w}_\theta(z,s))^\top\Sigma^{-1}(x-\hat{w}_\theta(z,s)) - \mathsf{KL}\big(q_\phi(z|w,s)\|p(z)\big)\right], \tag{3}$$

where the expectation is evaluated empirically over samples (w,s) drawn from the training set, along with latent representations sampled from the encoder, i.e., $z \sim q_\phi(z|w,s)$.

In the second stage, we introduce an adversarial discriminator $D_\varphi : \mathcal{W}\times\mathcal{S} \to \mathbb{R}$ that aims to distinguish between actual data sample pairs (w,s) and synthetic pairs produced by the VAE generative model. This discriminator is applied in the manner of WGAN [15], with Lipschitz continuity enforced by spectral normalization [16], in order

to improve the perceptual quality of the generative model. We also incorporate a regularization term that aims to constrain how much the fine-tuned decoder can vary, in order to maintain consistency with the frozen VAE encoder. The adversarial training loss of the second stage is given by

$$\min_{\theta} \max_{\varphi} \mathsf{E}\left[D_{\varphi}(w,s) - D_{\varphi}(\hat{w}_{\theta}(z,s),s)\right] + \lambda\|\theta - \theta_{\star}\|_2^2, \tag{4}$$

where $\lambda > 0$ controls the proximal regularization, $\theta_{\star}$ are the decoder weights obtained by VAE training in the first phase, and the latent representation is z sampled from its prior, i.e., $z \sim p(z) = \mathcal{N}(0, I)$.

While the proposed RAFT-VG architecture (see Fig. 1) has various hyperparameters (e.g., latent dimension, encoder-decoder layer sizes, and so forth) that can be tuned to promote good learning performance, the ultimate goal of connecting the RAFT-VG model to Modelica for building simulation dictated the careful selection of hyperparameter values to achieve this end. This careful selection manifests itself in a few notable ways. First, the latent dimension is chosen to be 8, which is significantly smaller than those commonly used in the machine learning literature (usually at least 32, but often 64 or 128). Second, the interconnection between layers needs to be relatively simple for the C-code to remain simple. For instance, operations that are now popular in deep learning, such as batch normalization, layer normalization, or convolution layers, are not included in an effort to keep the C-code straightforward. Third, the conditional variables s are embedded with small-sized embedding layers to minimize C-code complexity. Finally, there is only a need to implement the decoder of the RAFT-VG network in C for translation in Modelica because only the decoder unit is responsible for time-series generation once the RAFT-VG has been trained. This benefit arises as a natural consequence of the neural architecture design. As the requirement that generative learning capabilities in Modelica must be implemented in the form of a C interface imposes a wide variety of software constraints, the generative architecture and set of hyperparameters must be designed in consideration of the code complexity and memory storage, rather than focusing only on generative performance metrics.

4. Modelica Integration

Notations. The `typewriter` font is used here along with the dot notation to reference the syntax of the Modelica language, including the names of Modelica libraries, names of models, etc. Furthermore, the `typewriter` font is used to refer to the names of other software packages. In addition, the dot notation is used to specify hierarchy in object-oriented modeling. As an example, consider the model of a `building` containing a zone named `zon`, which itself is composed of a room named `roo`. To access a parameter value, for example, the constant convection coefficient for room-facing surfaces of opaque constructions, `hIntFixed`, the dot notation would be `building.zon.roo.hIntFixed`. Finally, syntax highlighting in code listings is that used in the Modelica Language Specification[1].

There are several aspects to consider when integrating decoder-only time-series generative networks (TSGNs) like the RAFT-VG model into building simulations: 1) modeling, 2) model integration with building models, and 3) simulation workflow automation. Before describing the aspects considered, a brief summary of existing options is provided. Although explicit modeling of neural networks (NN) in Modelica has been attempted in the past [17], this approach lacks the ability to integrate with the major ML platforms for NN design, such as PyTorch, only supports a few NN architectures, and the extant library has not been updated to function with the current version of the Modelica language and the Modelica Standard Library (MSL). Recent efforts have also been made to integrate NN with physics-based simulators, including the use of the Functional Mock-Up Interface standard [18] to exchange the trained NN model with other frameworks [19], and the development of specialized Modelica libraries that support the use of trained NN models [20, 21].

The FMI standard was not employed in these investigations to avoid the inherent drawbacks that co-simulation has on simulation performance [22]. The `SmartInt` [20] library allows the use of TensorFlow TFLite models [20] and the `NeuralNet` library supports the Open Neural Network Exchange (ONNX) format [21]. Both `SmartInt` and `NeuralNet` libraries are attractive for their intended scope; however, the implementation of each of these libraries

[1] See Ch. 1.4 Notation in the Modelica Language Specification: `https://specification.modelica.org/master/introduction1.html`.

has drawbacks. For this work, `SmartInt` was deemed unsuitable due to its lack of support for `PyTorch` and the added complexity caused by its dependencies. In comparison, the `NeuralNet` library is only available for use in the Wolfram SystemModeler Modelica tool, which is not able to compile models from the `Buildings` library that form a core motivation of this work. More importantly, the `NeuralNet` library requires the ONNX Runtime library and its C API, which adds not only complexity but also overhead in software integration. Regardless, one commonality of both libraries is how they leverage the standardized external function interface specified in the Modelica Language Specification (see [23, Ch. 12.9]), which supports external functions written in C and others. This is an attractive feature that could be exploited to pursue a different approach that meets the requirements for this work.

The goal of the software integration approach used here was to minimize all dependencies (only for simulation purposes) on external software tools other than the C compiler and the Modelica tool, in this case Dymola [24, 25], which itself requires a C compiler. The standardized external function feature in the Modelica language allows the integration of C-based TSGN models with the Modelica simulation models. Meanwhile, simulation workflow automation can be achieved by interfacing the simulator executable, which contains both the TSGN and the building models, with a suitable scripting tool supported by Dymola. To achieve this goal, the requirements for the three aspects considered at the beginning of this section are thus specified while a schematic diagram that illustrates the integration and implementation carried out to meet those requirements is shown in Fig. 2.

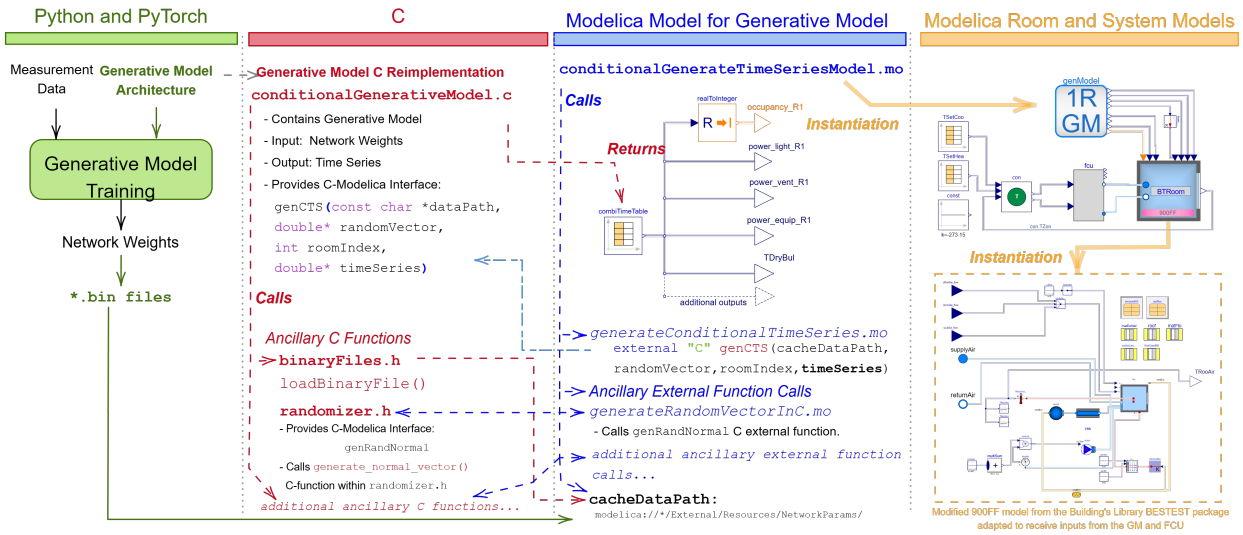

Figure 2: Integration of a Deep Generative Network with a Single-Zone Thermal Model of a Building

The first aspect to consider is that the TSGN models must be modeled in such a way that they can interact with the building simulation model. The TSGN models and their integration should meet the following requirements:

R1. Requirement 1: Simulation of the TSGN model needs to be integrated with the simulation model and must be computationally efficient. The burden of simulating the TSGN models along with the building model should be modest or negligible compared to the building model alone. This is a key requirement, as simulation models are complex and themselves have a high computational burden [26].

R2. Requirement 2: TSGN models should be easily populated with training data generated using PyTorch.

R3. Requirement 3: The parameters of the trained TSGN models should be exchanged in such a way that they are stored in file formats of as small a size as possible.

As Fig. 2 shows, R1 is met by implementing the TSGN model architecture in C; therefore, the model will be integrated into the simulation's source code and compiled into the executable `dymosim.exe` in Dymola. To meet R2, the TSGN model architecture is also defined in PyTorch where the occupancy, equipment, and lightning loads

data is ingested and used to train the NN. After the TSGN model parameters are obtained, they are stored in binary files (`*.bin`) in Fig. 2. As the building models are implemented in the Modelica language, it is necessary to integrate the TSGN models with them. To interface them the following considerations are made:

R4. Requirement 4: The TSGN model should be integrated with the building simulation model through a customized `block`. The inputs to the TSGN model should be passed through the Modelica model, and outputs of the TSGN should be interfaced through `RealOutput` or `IntegerOutput` blocks from the MSL to drive occupancy, equipment, and lighting loads.

R5. Requirement 5: The output of the TSGN model should provide a time series that populates a `CombiTimeTable` from the MSL that contains the values of the predicted variables at 15 minute intervals. These values should be of real or integer type, depending on the variable. The lookup table needs to apply a sample-and-hold and periodic extrapolation outside the range for which is defined to each variable. These functionalities should be within the `block` defined in R4.

R6. Requirement 6: The `block` from R4 also needs to be able to read the NN parameter data and pass them to the NN model to obtain the predicted time series. Reading these parameters should be fast and efficient.

Referring again to Fig. 2, R4 and R5 are met by calling different C functions that provide input to the TSGN model and request its output to populate the lookup table. This is illustrated for one of the C functions: `generateConditionalTimeSer` that is called in Line 5 of Listing 1. The function itself is defined through the `external` function as shown in Listing 2 which provides the TSGN output for the variable of real value. After being executed during the initialization of the model, `generateConditionalTimeSeries` in Listing 1 provides the `timeSeries1` output that will pass its data to `lookUpTableValues` in Line 6. Next, in Line 7, `CombiTimeTable` is instantiated and data are provided through the `lookUpTableValues` parameter. Moreover, the sample-and-hold and periodic extrapolation are defined by the `smoothness=...` and `extrapolation=...` modifiers, while the `timeScale=...` modifier defines the TSGN output period of 15 minutes. Finally, in Line 11, the `light_R1` instantiates a `RealOutput` interface that is coupled in Line 15 to the corresponding output of the table, i.e. `CombiTimeTable.y[2]`.

Listing 1: Excerpt of the Source Code Conditional Time Series Block Calling the GAN

```
1  model ConditionalGenerativeTSModel
2  import MB = Modelica.Blocks
3  constant String cacheDataPath=Modelica.Utilities.Files.loadResource("modelica://
       GenerativeModelForBuildSim/Common/External/Resources/NetworkParams/");
4  ... (many lines ommitted)
5  parameter Real[nSamplesPerDay*nSignals] timeSeries1 = generateConditionalTimeSeries(cacheDataPath,
       randomVector1, roomIndex1);
6  parameter Real[:,:] lookUpTableValues = [tix, timeSeries1Reshaped, ...];
7  MB.Sources.CombiTimeTable combiTimeTable(table=lookUpTableValues,
8  smoothness = MB.Types.Smoothness.ConstantSegments,
9  extrapolation = MB.Types.Extrapolation.Periodic,
10 timeScale=24*3600/nSamplesPerDay)
11 Modelica.Blocks.Interfaces.RealOutput light_R1;
12 ... (many more lines ommitted)
13 equation
14 ... (many connect statmenets ommited)
15 connect(light_R1, combiTimeTable.y[2]);
16 ... (many connect statmenets ommited)
17 end ConditionalGenerativeTSModel;
```

In the case of R6, it can be observed in both Listings 1 and 2 that the string `cacheDataPath` points to a specific directory where the `*.bin` files are located, and the files are read by another C routine within the `external genCTS` function on Line 6 of Listing 2. It should be noted in this Listing that the `annotation` points Dymola to the location of the C function `genCTS` so that it can be included as part of the integrated simulator code. In addition, some additional functionalities are included to obtain predictions from the NN and shape them into the expected input of the `combiTimeTable`. The NN must be initialized with a random vector, and its output must be reshaped in the form expected in R5. This is also achieved through C code mapped to other external Modelica functions that are omitted here, due to the lack of space.

To generate the requisite normally distributed random vectors, we employed the Box-Muller transform, which is straightforward to implement in C as it involves simple arithmetical operations; see Listing 3. The Box-Muller

method transforms two uniformly distributed random variables ξ_1, ξ_2 into two independent standard normal random variables $\hat{\xi}_0, \hat{\xi}_1$. The transformation is given by: $\hat{\xi}_0 = \sqrt{-2\log(\xi_1)}\cos(2\pi\xi_2)$ and $\hat{\xi}_1 = \sqrt{-2\log(\xi_1)}\sin(2\pi\xi_2)$. In our implementation, only $\hat{\xi}_0$ is used to generate a standard normal random variable. The variables ξ_1 and ξ_2 are uniformly distributed between 0 and 1, and are obtained by normalizing the output of the C standard library function `rand()`, which generates pseudo-random integers. The relationship is expressed as:

$$\xi_1 = \frac{\text{rand()}}{\text{RAND\_MAX}}, \quad \xi_2 = \frac{\text{rand()}}{\text{RAND\_MAX}}.$$

The function `generate_random_normal()` implements this transformation to return a scalar standard normal random variable $\hat{\xi}_0$. To generate a Gaussian-distributed random vector χ with a specified mean μ and standard deviation σ, we compute the i-th element $\chi_i = \mu + \sigma\hat{\xi}_0$, where $\hat{\xi}_0$ is the output of the `generate_random_normal()` function. By iterating over i, the vector χ is filled in.

Listing 2: External Function in Modelica Linking the Output of the GAN

```
function generateConditionalTimeSeries
input String cacheDataPath = Modelica.Utilities.Files.loadResource("modelica://.../Resources/
    NetworkParams/");
input Real[16] randomVector = {-1.1258, ...};
input Integer roomIndex = 2; // 0, 1, or 2
output Real[384] timeSeries;
external "C" genCTS(cacheDataPath, randomVector, roomIndex, timeSeries)
annotation (
IncludeDirectory = "modelica://.../Resources",
Include="#include \"conditionalGenerativeModel.c\"");
end generateConditionalTimeSeries;
```

Listing 3: C-Code for generating random vectors as inputs to the generative model

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define M_PI acos(-1)

/* Box-Muller transformation to generate Gaussian random number */
double generate_random_normal() {
        double u1 = rand() / (double)RAND_MAX;
        double u2 = rand() / (double)RAND_MAX;
        double z0 = sqrt(-2.0 * log(u1)) * cos(2.0 * M_PI * u2); // Box-Muller
        return z0;
}

/* Function to generate Gaussian random vector */
double* generate_normal_vector(double mu, double sigma, int seed, int vectorSize) {
        srand(seed);
        double* vector = (double *)malloc(sizeof(double) * vectorSize);
        for (int i = 0; i < vectorSize; i++)
        vector[i] = mu + sigma * generate_random_normal();
        return vector;
}
```

Finally, one of the available scripting interfaces provided by Dymola must be selected to execute multiple simulations automatically. Among the various interfaces, the most attractive is the Python Interface for Dymola [25], an API for executing Dymola commands using a Python program. This choice was made because PyTorch is already being used to train NN models. Using this interface, the model parameters, weather data files, etc., can be specified and used for specific simulation cases. For example, in Fig. 2 the `combiTimeTable` of the `ConditionalGenerativeTSModel` requires the parameter `roomIndex1` as shown in Line 5 of Listing 1. Hence, using Python, it is possible to change the value of this and any other parameter through Dymola's Python API, and then ask Dymola to run the simulation (i.e., to run the TSGN and building model together for a specific room). To speed up this process, the model is first translated (see Chapter 1.3 of the Modelica Specification) using the `translate` command of the Dymola Python API, which generates the code of the simulator that can simulate the model. Thus, every time that parameters are changed within a loop, the model does not need to be translated; i.e., code generation is avoided, reducing time.

Details of the implementation of batch simulation, processing and analysis Python scripts and utility functions are omitted here; however, the reader is pointed to reference examples from [27], which describes how to use Python

interfaces from both Dymola and OpenModelica for similar purposes.

5. Results and Discussion

5.1. Description of Modelica Building Simulation Model

We developed a high-fidelity building emulator using the Modelica Buildings Library 10.0.0[2]. The emulators are based on the ASHRAE BESTEST model, which are benchmark models for building energy simulation with open-sourced information such as dimensions and constructions [28]. The building represents a single zone with a window on the south wall and a constant infiltration mass flow rate. There are two variations of construction, lightweight case 600FF and heavyweight case 900FF. The exterior walls and roof of cases 600FF and 900FF are plaster board with fiberglass insulation and concrete block with foam insulation, respectively. The floor of case 600FF is timber construction, and the floor of case 900FF is a concrete slab. The external disturbances are characterized by the typical meteorological year data for San Francisco, USA.

The emulators include a simple fan coil unit (FCU) to condition the room that is regulated by a proportional-integral (PI) dual-setpoint controller to maintain the room temperature within the heating and cooling references (setpoints). When the FCU is activated, the supply fan runs at a constant speed to circulate the air through the heating and cooling coils. The heating and cooling setpoints are converted to the supply air temperature setpoint by the PI controller, and the coils are activated to reach the setpoint. The conditioned air is then supplied to the room, where the air is assumed to be well-mixed. As this study focuses on the control performance, the energy impact of the FCU is simplified. The electric heating coil has a constant efficiency of 0.9, and the cooling coil operates at a constant coefficient of performance (COP) of 3.0.

The internal disturbances are defined to represent a small office with a floor area of 48 m^2, where the operating hours are assumed to be from 8 AM to 6 PM and at most 15 people are assumed to be in the zone, with usually no one in the office outside office hours, as seen from the data. The zone fills up around 8AM, on average to 7 people around 9AM, and remains around that level of occupancy except at lunchtime (noon) where the average drops to around 2 people. The sensible heat gain density of the office equipment is set to be 5 W/m^2 with the radiative heat gain comprising 30% of the total, which is a typical light office load. Similarly, the lighting heat gain is assumed to be 10 W/m^2 with a 50% radiative fraction. The occupant-related activity generates both sensible and latent gains that are involved in the room heat and moisture mass balances. These gains, which correspond to moderate office work, are assumed to be 73 $\mathrm{W/person}$ sensible with a 60% radiative fraction, as well as a 45 $\mathrm{W/person}$ latent gain. These heat gains are active during the occupied period and zero during unoccupied periods. The heating and cooling occupied and unoccupied setpoint temperatures are (18°C, 20°C) and (14°C, 28°C).

5.2. Data Collection

For the purpose of training generative models of internal disturbances, we use real experimental data collected from SUSTIE, which is a next-generation commercial office building located in Japan. The name SUSTIE combines the words "Sustainability" and "Energy" and the building is designed to research and demonstrate energy savings and workers' health and comfort. The four-story SUSTIE building has a total floor area of approximately 6456 m^2 which includes nine office spaces that are regularly used by around 260 office workers, an open-feel atrium area, a cafeteria, and a gym. SUSTIE's building management system collects data on electrical energy consumption, meteorological and indoor environment conditions, occupancy, and equipment operational data to analyze and control energy consumption and comfort during building operations. The electrical energy consumption is measured separately for different types of equipment (air-conditioning, ventilation, lighting, hot water supply and elevators) and for each room. The occupancy, i.e., the number of people in each room, is counted by the access control system using card readers installed in each area.

A dataset collected at SUSTIE over consecutive 20 months from January 2021 to August 2022 is used in this work for generative modeling. We adopted a number of data pre-processing steps as described below to make this dataset tractable and usable for training models. Any missing values in data signals were filled using linear interpolation, and

[2]https://simulationresearch.lbl.gov/modelica/

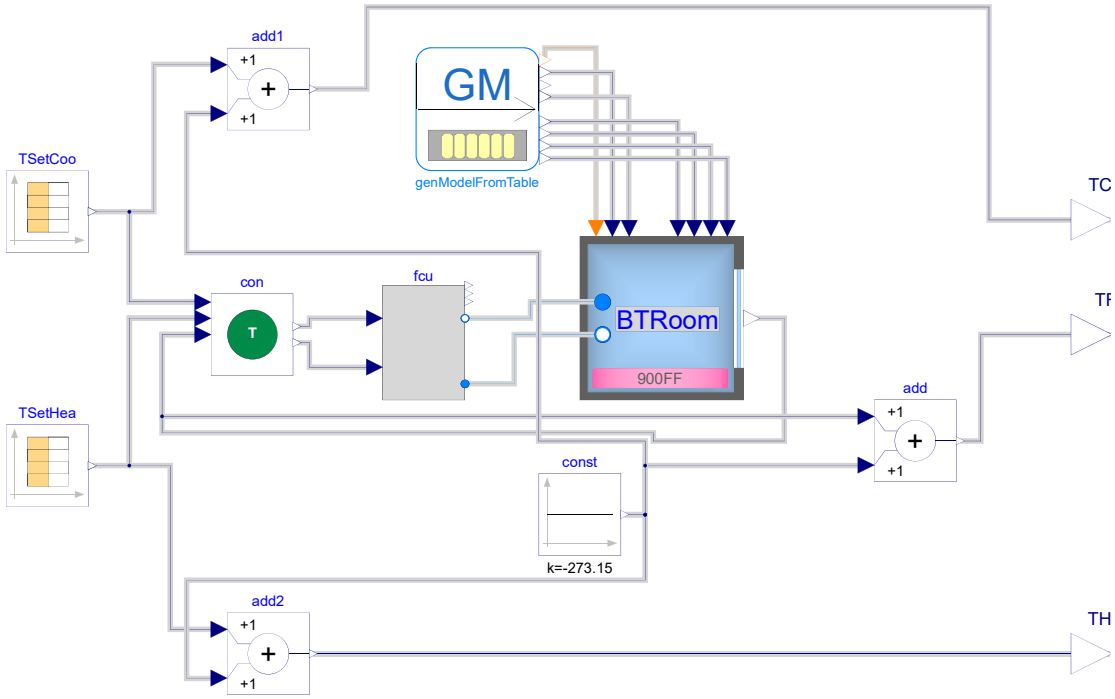

Figure 3: Modelica interface of generative model (`GM`) block to the building control algorithm (`con`), the actuator fan coil unit (`fcu`), and the thermal dynamical model (`BTRoom`); here we show BESTEST Case 900FF. Inputs to the model include setpoint references for cooling and heating, which define the temperature comfort constraints. The primary output is the simulated room temperature.

all data signals were synchronized to identical sampling times with a sampling rate of 1 minute. The electrical energy consumption (kW·h) signals of different equipment were also converted to power consumption (W) by assuming piece-wise constant power signals between two successive energy measurements.

Noting that time constants for many internal disturbances of a building are typically of the order of several minutes, we downsampled the dataset with a sampling rate of 15 minutes. Since the multi-zone SUSTIE building is much more complex than a single-zone building simulator described in the previous section, we select a proportionate subset of signals from the SUSTIE dataset for our modeling efforts. In particular, we consider a set of 12 signals which include the power consumption by HVAC equipment, ventilation, lighting systems and the occupancy in three different office spaces located on the same floor of SUSTIE. While this reduction in the size of dataset also makes the training process more tractable, we emphasize that the scaled dataset still maintains characteristic day-to-day usage trends of a real commercial office building.

5.3. RAFT-VG Generative Network Implementation

We train the conditional RAFT-VG by first scaling the data by their median orders of magnitude; the temperature signals in °C are scaled down by 10, whereas power signals in W are scaled down by 1000 or 100. Note that all the time-series to be generated in this work are non-negative. The data tensor is of size $600 \times 96 \times 12$, since the dataset consists of 4 signals in 3 rooms collected every 15 min for 600 days. Two conditional inputs are considered: (i) the *zone number*, as the data is for 3 distinct zones in the SUSTIE building, and (ii) whether the scenario to be generated is a *workday* or a *holiday*, which greatly affects occupancy, and therefore usage. Note that even though

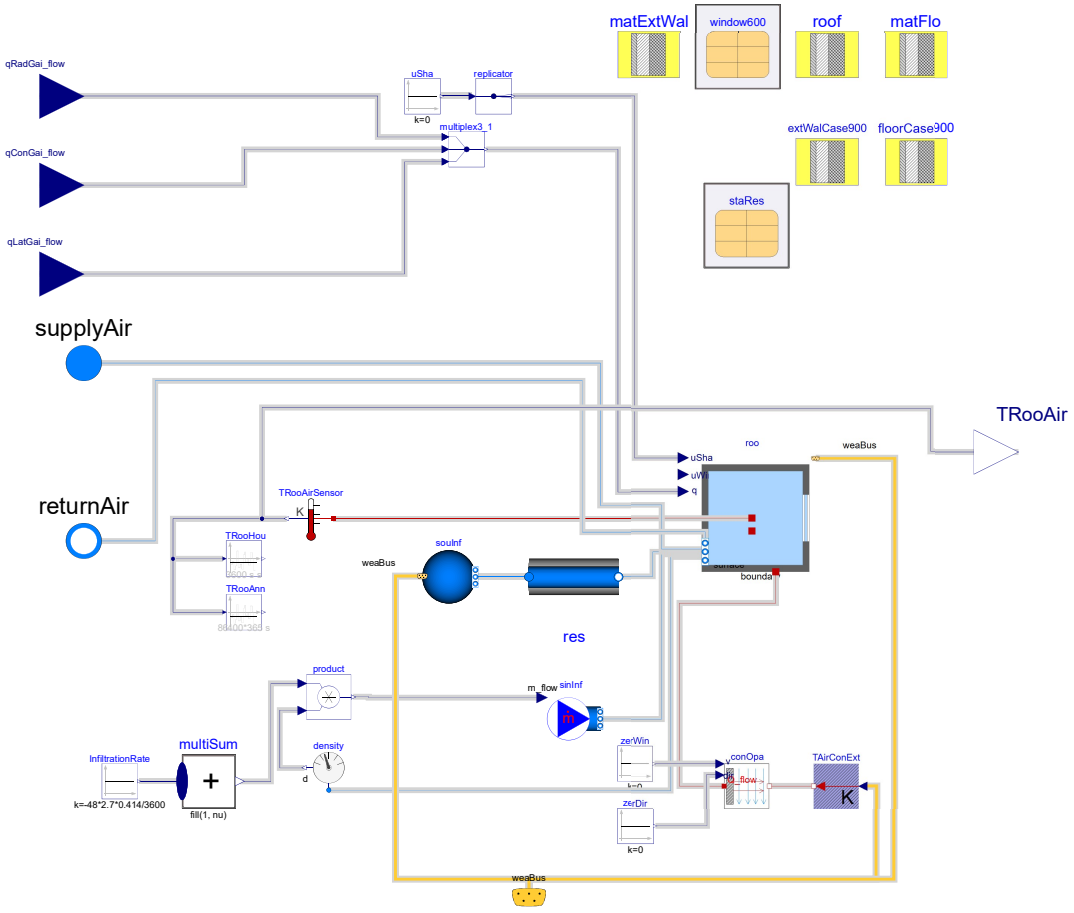

Figure 4: Existing building thermal dynamical model in Modelica BESTEST Library. Our generative network interacts with this module via the heat load components and the weather bus.

we take care to include *known* national holidays in the holiday conditional data, there may be days in the dataset that are *unknown* holidays. For example: holidays specific to the building residents such as company-specific or family-specific holidays or sudden holidays given due to extreme weather conditions. These days will not be possible to label correctly as a 'holiday' if it falls on a workday, without some customized data mining solution. These unknown holidays will instead contribute to the 'workday'-conditioned dataset, which will subsequently contribute to some low occupancy during workdays.

For ease of conditional RAFT-VG training, the data tensor is reshaped to $1800 \times 96 \times 4$ so each sample is a scenario agnostic of the room, and the conditional inputs are of size 1800×2. For the training itself, we randomly split the data into a training set with 1500 samples and a hold-out validation set with the 300 remaining samples. The training loss is computed with the two losses (3) and (4), and for the validation loss, we compute the symmetric Jensen-Shannon divergence[3] (JSD) between the validation data distribution and the decoder-generated distribution. The JSD is computed efficiently by approximating both by their mean and variance, and assuming independence and no cross-correlation. Network weights are saved when the total JSD over the validation data batches is lower than the

[3]Assuming both n-dimensional distributions are Gaussian, with means μ_P and μ_Q, and covariance matrices Σ_P and Σ_Q, the Jensen-Shannon divergence between the Gaussian distributions P and Q can be expressed as: $\mathsf{JSD}(P \parallel Q) = \frac{1}{2}\mathsf{KL}(P \parallel M) + \frac{1}{2}\mathsf{KL}(Q \parallel M)$ where KL is the

JSD computed in all previous training iterations. The batch size is fixed at $n_{\mathsf{batch}} = 128$. We set $\beta_{\mathsf{VAE}} = 10^{-4}$ and the total number of VAE training iterations is 9000 with an Adam solver with learning rate 10^{-4} without scheduling. We allot 1000 iterations for GAN-based adversarial fine-tuning with learning rate 10^{-6} and $\lambda = 10^5$ to discourage finding solutions too far from the phase-1 decoder weights. The discriminator is optimized with an RMSProp solver and the decoder is updated with an Adam solver with momentum factors $(0.5, 0.999)$ instead of the default $(0.9, 0.999)$. For each outer loop decoder update in (4), the discriminator is updated 2 times, which is a common trick that improves Wasserstein-GAN training [15].

All three conditioning inputs are passed through an embedding layer using `torch.nn.Embedding` with an embedding dimension of 4, which is referred to as conditioning all three components of the RAFT-VG. Note that these layers are also learned during training. For the conditional encoder, the input 96×4 tensor is flattened and a batch has size $n_{\mathsf{batch}} \times 384$. This is passed through 5 fully-connected encoder layers activated by `LeakyReLU` functions, with hidden dimension $(256, 256, 256, 128, 64)$ and ending with a latent distribution in $n_z = 8$ dimensions. A low latent dimension was chosen specifically because we will search this latent domain for representative scenarios. The encoder has two outputs: the mean and log-variance of the encoding. These are then passed through a standard reparameterization operation and sampled using an isotropic standard Gaussian distribution which is fed to a decoder.

Since the decoder is arguably the most important component of our architecture, as this is the component that generates the scenarios, we investigated various architectures and activation functions for its design. For instance, because our signals are always non-negative, we ensure that the final layer is passed through a non-negative activation function. As the ReLU function in the output does not work well in practice because it leads to sudden drops to zero in the generated scenarios, our final layer is passed through a smooth softmax function. We also discovered that using ELU functions as activation functions in the intermediate layers of the decoder improves the smoothness of the generated scenarios, as opposed to ReLUs or LeakyReLUs. The decoder has 5 fully connected hidden layers, with sizes $(64, 64, 128, 128, 256)$ and an output of size 384 to match the encoder's input. The input to the decoder is an 8-dimensional latent along with the $4 + 4 = 8$ embedded conditional inputs. While this is not part of the training procedure, if the output of a decoder channel needs to be an integer, for example, an occupancy number, then we can perform a post-hoc operation such as rounding to attain that. This can be done directly as a PyTorch operation in the decoder at inference, or by a Dymola operation (see R4 in Section 4) during translation of the decoder into Modelica.

The conditional WGAN has 4 connected layers, each of which have spectral normalization, and are subsequently activated by `LeakyReLU(0.01)` functions followed by `Dropout(0.1)` for regularization during phase-2 training of the RAFT-VG. The 384-size input to the discriminator is conditioned with the 8-dimensional conditional embedding, and the output is a scalar. The final layer of the WGAN does not have dropout or spectral norm regularization, and is a linear layer without activation.

5.4. Effect of Conditioning Inputs on Generated Profiles

We demonstrate the effect of conditioning inputs (Workday/July, and Workday/December are the conditional input pairs) on the synthetic data generated by conditioned RAFT-VG via Fig. 5 and Fig. 6. The synthetic data distribution for 24h is compared against the true data distribution by sampling the latent space 100 times with the same conditional input, and plotting the median scenario (solid line) along with the 5-95 confidence intervals (lighter shading). Each figure contains four subplots that depict 24-hour variations in occupancy, relative humidity, light power, and solar radiation scattering, where the true data is represented by the color blue, and the synthetic data is represented by the color orange.

In terms of occupancy, both figures show high values during standard working hours, tapering off in the evening and nighttime. The RAFT-VG also captures the emptying of the zone at lunchtime, which is evident from the sudden dip in the median around noon. Note that the true data has discrete occupancy values, whereas the approximated distribution is continuous (we revert back to integers by rounding, post-hoc). Overall, the approximate distribution is close to the true one for both summer and winter, and captures key trends.

Kullback-Leibler divergence, which has the closed-form

$$2\mathsf{KL}(P\,\|\,Q) = \mathsf{Tr}(\Sigma_Q^{-1}\Sigma_P) + \|\mu_Q - \mu_P\|^2_{\Sigma_Q^{-1}} - n + \log\left(\frac{\det(\Sigma_Q)}{\det(\Sigma_P)}\right)$$

with M as the average distribution $\frac{1}{2}(P + Q)$.

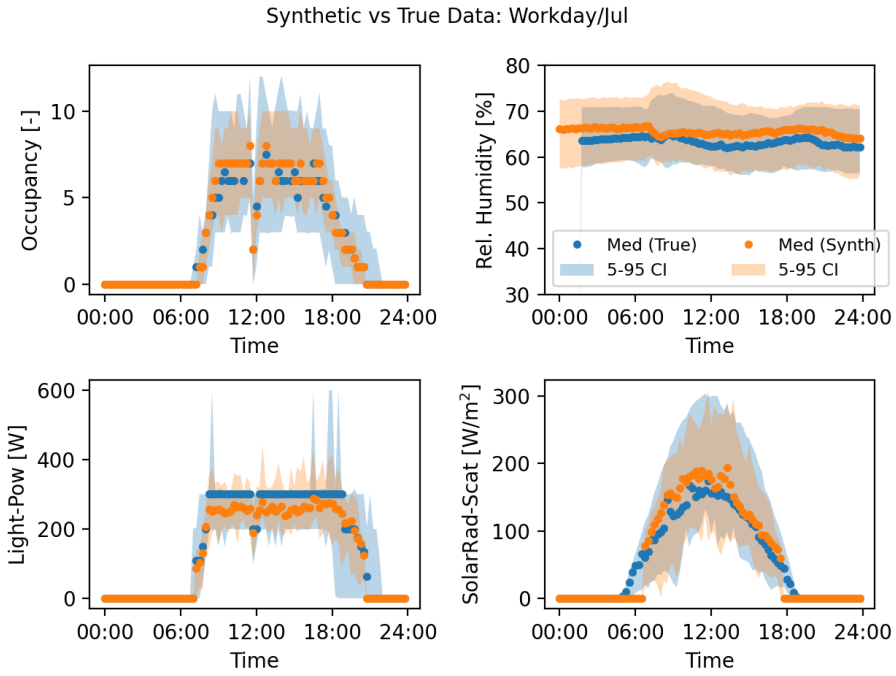

Figure 5: Comparison of 24h distribution of occupancy, relative humidity, lighting power, and solar radiation on a workday in summer. The blue/orange dots indicate median samples every hour for the true/synthetic data, and the faded bands around the median are the 5-95 percentile confidence intervals.

Relative humidity and scattered solar radiation plots also demonstrate similar accuracy in terms of distributional learning, and the effect of the conditional input is clear. In summer, the relative humidity distribution is over 65%, whereas it is <50% in the winter: this has been automatically learned by the generative network. Furthermore, as expected, the scattered solar radiation in the summer is lower than the winter, and the total daylight hours of the summer is longer than in winter.

Light power usage demonstrates clear seasonal variations in both figures. In December (Figure 6), light power peaks in the early morning and late afternoon, consistent with shorter daylight hours. The true data features more erratic peaks, while the synthetic data appears smoother. In July (Figure 5), the synthetic and true data align more closely, as the longer daylight hours result in lower overall light usage, with both datasets showing less deviation.

While the statistics have also been captured relatively accurately, there are some signals where the confidence intervals are over/underestimates. In particular, lighting power in the winter has a confidence interval in the true data that is quite high in the afternoon, but this is not captured very well by the generative network because of inherent smoothing effects due to convolutional layers in the decoder. Such smoothing cannot be simply removed however, because some signals (e.g., humidity and temperature) should not have high-frequency effects during generation. One could customize the channels where the convolution layers are implemented to alleviate this issue.

5.5. Effect of Varying the Latent Distribution σ

Figure 7 illustrates the variation of occupancy over time with different levels of uncertainty in the synthetic data generation process, controlled by the parameter σ in the latent distribution of the RAFT-VG network. The median of the true occupancy data is represented by the blue line, along with a shaded region indicating the 5-95% confidence interval. The synthetic data is shown for three different values of σ: $\sigma = 1$ (orange), $\sigma = 2$ (green), and $\sigma = 5$ (red). Each of these synthetic datasets is also accompanied by a shaded region that represents the corresponding confidence intervals. As σ increases, the confidence intervals become progressively wider, reflecting greater variability in the synthetic data.

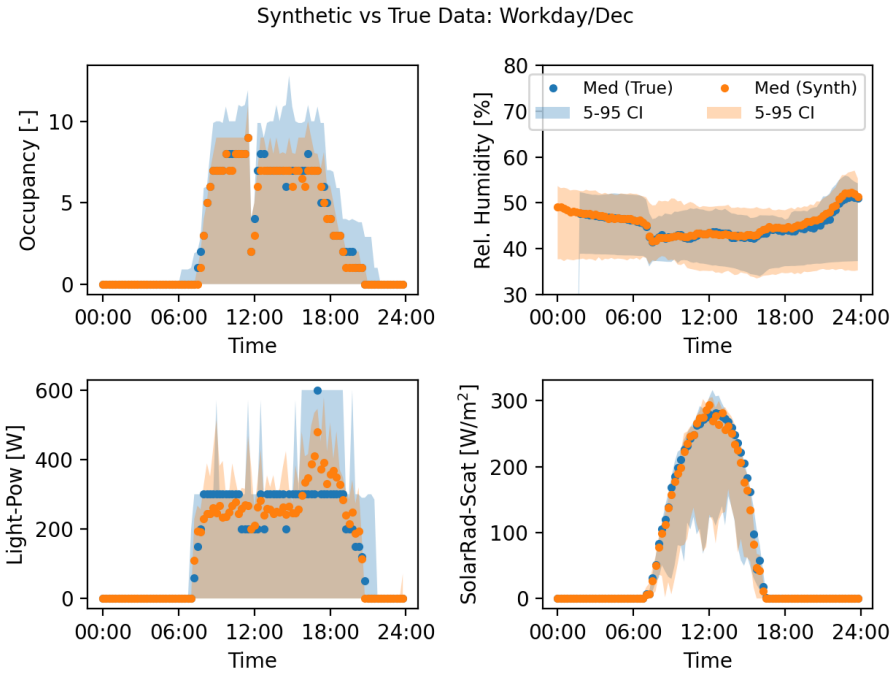

Figure 6: Comparison of 24h distribution of occupancy, relative humidity, lighting power, and solar radiation on a workday in winter. The blue/orange dots indicate median samples every hour for the true/synthetic data, and the faded bands around the median are the 5-95 percentile confidence intervals.

We also note that the synthetic data medians align relatively well with the true data. This is as we would expect, since the latent distribution is zero-mean Gaussian so the "average" (similar to the median) of the synthesized data should be close to the true trend. Lower σ values provide more reliable synthetic data, with narrower confidence intervals that closely mirror the true data. In contrast, higher σ values introduce greater variability, which can be useful for exploring a broader range of possible outcomes. Such behavior can be useful in a variety of cases: for example, the scenarios generated for $\sigma = 5$ sometimes have occupants in the zones late at night, which is not reflective of the real data, but automatically generates a few "unlikely" scenarios that are very useful for analyzing the behavior and performance of HVAC in unusual situations.

5.6. Comparison with state-of-the-art time series generative networks

Herein, we demonstrate some advantages RAFT-VG has over state-of-the-art TSGNs. We perform a set of experiments wherein the latent dimension is changed, and the network is trained, upon which its performance is evaluated. The results are reported in Table 1, with each TSGN name followed up by the latent dimension used. The metrics and models we use as competitors are in accordance with existing methods reported in TSGN such as [10]. The performance indicators include a deterministic fit metric: the normalized root-mean-squared error (NRMSE) between the median of the true and generated distributions; and two distributional fit metrics, the Kullback-Liebler (KL) and Jensen-Shannon (JS) divergences. All metrics were computed channel-by-channel of the multivariate time-series and then averaging the channel-wise scores. The NRMSE shows how well the median is captured by the corresponding TSGN, and it is immediately clear that the RAFT-VG and VAE models are outperforming the rest, especially at low latent dimensions. However, in terms of the fidelity of the generated distribution, RAFT-VG has improved performance overall, e.g. as per the JSD metric. Furthermore, and as expected, severely compressing the latent vector does result in slight worsening of performance e.g. RAFT-VG-32/128 are both better than RAFT-VG-8, but because the network architecture is simpler compared to DoppelGANger, the encoding-decoding transformations trade-off overfitting with expressivity and are able to reproduce the true distribution well even with $n_z = 8$. To summarize, from Table 1, there is encouraging evidence that the RAFT-VG model not only performs comparably to the existing state-of-the-art —

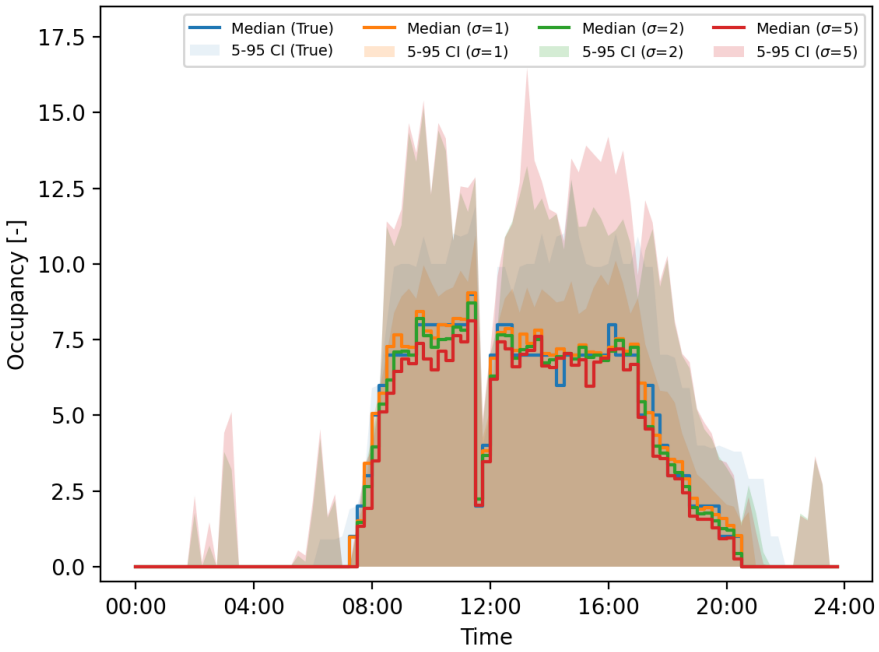

Figure 7: Distribution over 24h of synthetic data generated by varying the standard deviation of the latent distribution of the generative network. The true data median and confidence intervals are in blue, while increasing $\sigma \in \{1, 2, 5\}$ values are in orange, green, and red, respectively.

a trait the VAE architecture itself seems to share, with VAE-8 performing quite well also — but that compressing the latent dimension for assisting downstream implementation in Dymola does not cripple the network performance, making it a good choice for TSGN in building simulation.

TIME-SERIES GENERATIVE NETWORK	NRMSE	KLD	JSD
DoppelGANger-8	0.132	0.419	0.044
DoppelGANger-128	0.107	0.259	0.022
VAE-GAN-128	0.244	0.556	0.119
RAFT-VG-8	0.029	0.155	0.010
RAFT-VG-32	0.023	0.141	0.005
RAFT-VG-128	0.024	0.140	0.005
W-GAN-128	0.114	0.994	0.133
VAE-8	0.039	0.204	0.025
VAE-128	0.033	0.166	0.019

Table 1: Performance metrics for various models. The integer following the network name is the latent dimension used for training.

Since the VAE and RAFT-VG models perform similarly, we wanted to understand better why DoppelGANger showed slightly worse performance, despite being considered one of the leading TSGN models. The reason is made clear in Figure 8, which presents a comparison of real occupancy data against synthetic data generated by these two competitor models: RAFT-VG and DoppelGANger [29]. The real data is depicted by the blue dashed line, while the synthetic data generated by RAFT-VG and DoppelGANger are represented by the orange and green lines, respectively. Additionally, shaded regions around each synthetic line indicate the 5-95% confidence intervals for both models. The plot shows that both synthetic models capture the general trend of occupancy over the 24-hour period, with a clear peak

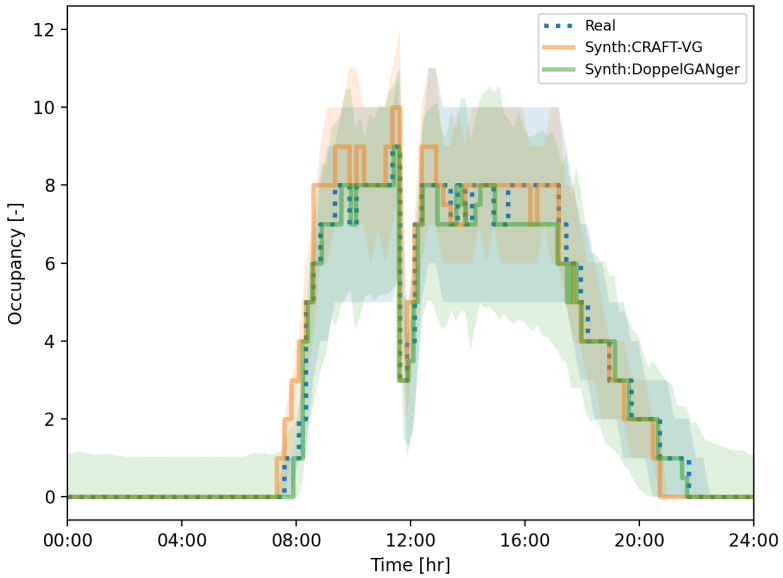

Figure 8: Comparison of learned distributions by our proposed method RAFT-VG-8 (orange), the state-of-the-art DoppelGANger-8 (green), and the true data distribution (blue). The conditional input provided is for a weekday in the summer.

during working hours, tapering off into the evening and reaching zero during off-hours. However, there are notable differences between the models. The RAFT-VG model closely follows the real data during peak occupancy hours, particularly between 8AM and 6PM, with narrower confidence intervals than DoppelGANger. This indicates that RAFT-VG generates more accurate and reliable synthetic data. In contrast, the DoppelGANger model shows greater deviation from the real data, particularly during transition periods when occupancy begins to rise in the morning and decrease in the afternoon. The wider confidence intervals for DoppelGANger is especially worrisome on the off-work hours such as before 8AM and after 6PM, since none of the true data has any occupants in this zone at this time, and therefore the non-zero confidence interval there is a hallucination by DoppelGANger. Our proposed approach with $\sigma = 1$ is deterministically zero outside of work hours as the data suggests. The spurious non-zero occupancies are likely due to DoppelGANger selecting lower and upper bounds for the signal based on latent sampling, which, while useful for many applications, results in a lower-fidelity distributional fit in cases when a deterministic trend is necessary to reproduce the signal; e.g. non-work-hour occupancy.

Latent generative models, such as RAFT-VG, offer substantial advantages over traditional models like Hidden Markov Models (HMMs), particularly when dealing with large datasets and multiple conditioning inputs. First, latent generative models are highly flexible and capable of learning complex, non-linear relationships within the data, which makes them particularly well-suited for high-dimensional datasets that do not conform to classical distributions (e.g., the Gaussian distribution). Additionally, latent generative models can incorporate multiple conditioning inputs with ease, providing a powerful framework for generating conditioned synthetic data across different scenarios. In contrast, HMMs and their variants are more limited in their capacity to handle such complexity. HMMs are designed to model sequences with discrete hidden states and rely on strong assumptions about the Markov property and transition probabilities. While effective for certain tasks, they struggle to capture the intricacies of large datasets with multiple covariates, often resulting in poorer model performance and less accurate synthetic data generation. If the underlying data is continuous-valued, then the HMM requires simplifying distributions to interface with a tractable training algorithm (e.g., Viterbi's method) that further detract from its performance on real-world data. Furthermore, HMMs scale poorly with increasing data complexity, which makes them less suitable for modern applications involving large, multi-dimensional datasets.

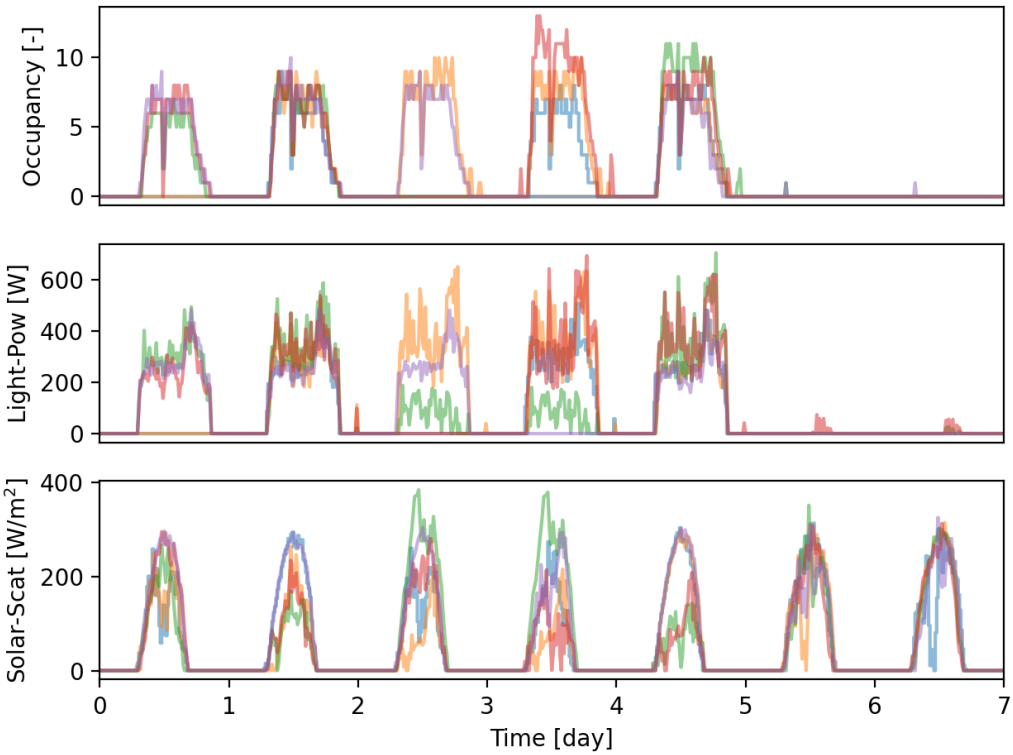

Figure 9: Illustration of multi-day scenarios being generated by RAFT-VG. The different colors are synthetic trajectories generated for 24h and stitched together for arbitrary lengths of time, e.g., a week in this plot.

5.7. Multi-Day Generation

Sequences of arbitary time length can also be generated via these tools. Rather than retraining RAFT-VG with the appropriate conditional inputs for these longer time series, which would effectively reduce the number of samples we have available, we generate and stitch together 24 hour scenarios with an appropriate selection of conditional inputs. Figure 9 illustrates this approach for generating occupancy, light power, and solar radiation time series over a week. Multi-colored lines indicate the different realization sampled and decoded using RAFT-VG. The conditional inputs for 7 days are $\{workday \times 5, holiday \times 2\}$ along with a month condition e.g., June. The occupancy for the last two days is nearly zero because these days are holidays. The variations between the different lines are also relatively minor, suggesting that the model consistently captures the general trends of occupancy over multiple days. However, some deviations occur, particularly during the transition periods between high and low occupancy, where the lines diverge slightly. This is indicative of the richness of the scenarios produced by the generative network. As such, the generate-and-stitch method can clearly generate scenarios for any arbitrary length of time.

5.8. Use-Case #1: HVAC Performance Monitoring

Figure 10 presents a comparative analysis of two building simulations in which the disturbance inputs are calculated either with a nominal model (nominal scenario) or by using the generated model described in this paper (generated scenario) for several key building performance parameters over a 350-hour period. The four subplots display time-series data for occupancy, room temperature (T_{room}), heating controller input, and cooling controller input. The black lines indicate outputs from the generated scenario, while the red lines indicate outputs from the nominal scenario. Additionally, the heating and cooling set points for T_{room} are shown in cyan in the second subplot.

In the first subplot, the occupancy data demonstrates cyclic patterns for both scenarios, with the occupancy peaking during working hours and falling to zero during off-hours. While the overall trend between the two scenarios is similar,

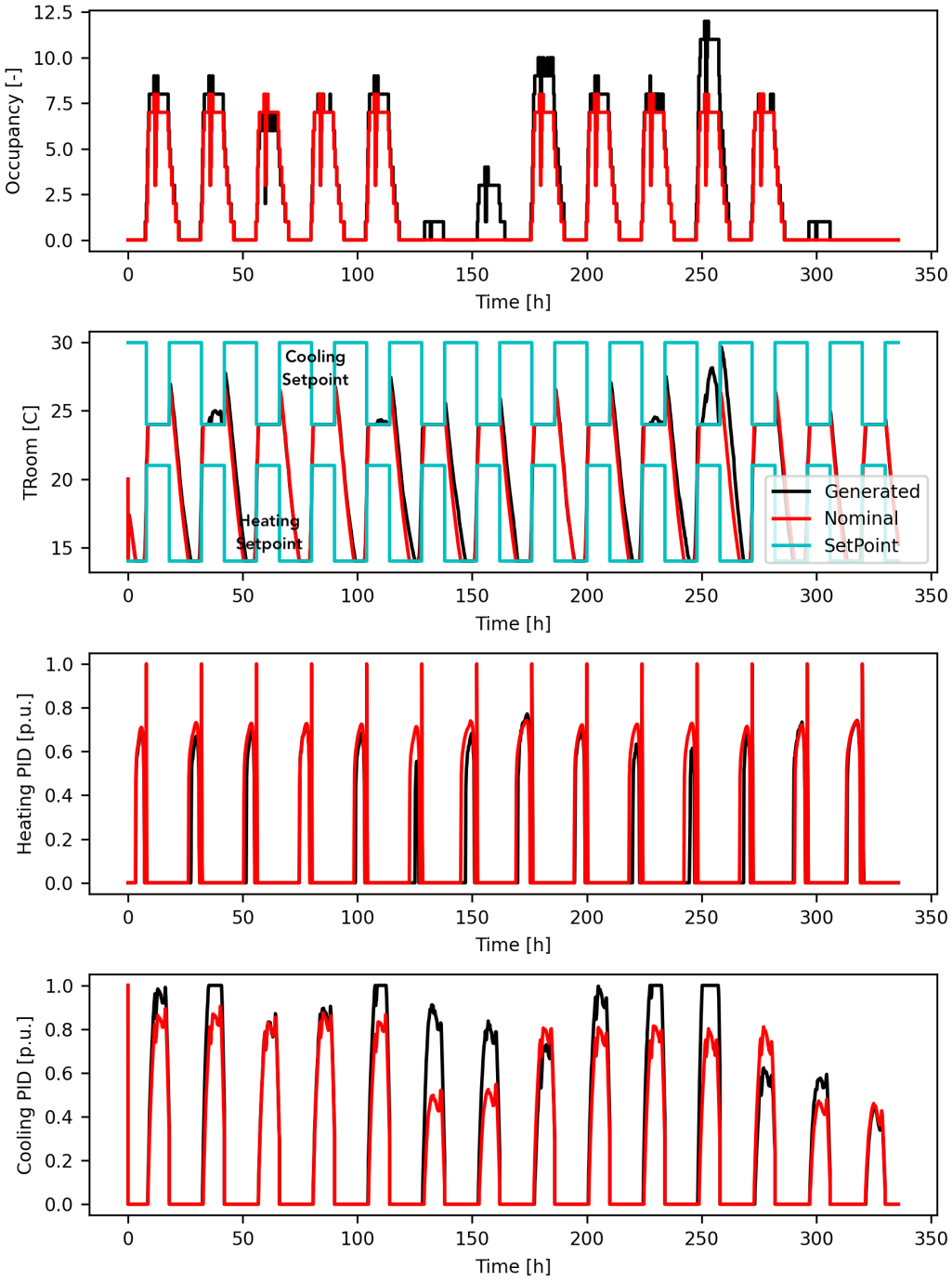

Figure 10: Closed-loop simulation of building with automatic control in Modelica using a nominal disturbance model and a generative disturbance model. Repetitive use of the nominal scenario over multiple days does not expose the inability of the HVAC system to follow temperature setpoint values under certain conditions around $t = 40$h and $t = 250$h, where the generative scenario makes these limitations clear. The two weeks are generated by conditioning the month to July.

discrepancies between the occupancy levels are evident, particularly between 150 and 200 hours and again beyond 250

hours. The varying occupancy levels from the generated scenario, in comparison to those of the nominal scenario, have a significant effect on the building performance because of differences in the room temperature control and HVAC load requirements.

The second subplot displays room temperature (T_{room}) from simulations of both scenarios, as well as the system setpoints. The data from the nominal scenario achieves tight control around the set points, maintaining temperature within acceptable bounds. However, the data from the generated scenario shows clear violations of the temperature constraints, particularly between 150 and 250 hours. During these periods, the room temperature in the generated scenario exceeds the set points, suggesting that the cooling system is undersized or inefficient in responding to the cooling demand. Conversely, periods of lower-than-expected temperature indicate that the heating system might be overcompensating or poorly sized for the thermal load. Observations of such violations that occur in realistically varying conditions, rather than at only one nominal disturbance profile, are critical to assess the designed system performance because they highlight potential issues in HVAC system sizing and suggest that the current system may be inadequate for maintaining thermal comfort.

Similar conclusions can be reached by examining the details of the third and fourth subplots, which illustrate the heating control and cooling control input signals, respectively. While the generated and nominal scenarios are qualitatively similar, the heating input signal in the generated scenario lags behind that of the nominal scenario, indicating that the heating system is not able to respond to the variations in the inputs that are captured the generated scenario. This delayed response may result in thermal discomfort for occupants, and indicate the that the heating system is undersized in response to occupancy-driven load changes. The generated scenario is also quite different from the nominal scenario during periods of high cooling demand, in which the HVAC system's cooling capacity is inadequate to meet the load and results in the room temperature violations visible in the second subplot. This serves to reinforce the implication that the cooling system's design capacity also may not have adequately accounted for peak loads, and that sizing adjustments may be needed to consistently maintain thermal comfort.

Overall, Figure 10 reveals several critical periods where temperature constraints are violated in both heating and cooling operation. These violations highlight the potential undersizing of the HVAC system, particularly in its ability to cope with fluctuating occupancy and external conditions. Information about such constraint violations could be used to improve the HVAC system design to ensure that it is sufficient to meet the peak loads without sacrificing occupant comfort. Studies such as [30] and [31] emphasize the importance of aligning HVAC system capacity with dynamic load requirements to properly size the equipment. The above analysis based upon realistic disturbance models reinforces this recommendation, as varying occupancy patterns and environmental conditions must be taken into account to prevent temperature constraint violations and ensure desired performance.

5.9. Use Case #2: Identifying Limiting Scenarios via Bayesian Latent Sampling

The latent generative models developed in this work may also be used to identify "limiting scenarios" representing disturbance sequences that lead to substantially different closed-loop performance. For example, a system designer might be interested in identifying a minimal number of simulations that would expose shortcomings in the equipment design, rather than relying upon an extensive exploration of the design space that requires extremely large numbers of simulations. Unlike context-based generators, these generative models learn a compact, low-dimensional latent space that can reliably reproduce long-horizon scenarios. This compactness enables the formulation of tractable optimization problems within the latent space, making it easier to identify such critical disturbance scenarios in a computationally efficient manner. Moreover, since these scenarios are generated from real data and are independent of the underlying simulation model, they allow for flexible assessment across different building configurations. As it would be challenging to manually identify the boundaries of such behavioral extremes, we develop a systematic (automated) sampling procedure that will generate these scenarios and thereby enable significantly improved performance monitoring capabilities.

In this work, we seek to identify a set of two limiting disturbance scenarios that yield distinctly different behaviors that can be applied to two building models, as detailed in Section 5.1. The first building model, described by the BESTest case 680, incorporates low thermal mass and high air infiltration (0.41 ACH), while the second building model, described by BESTest case 900, has high thermal mass and low air infiltration (0.041 ACH). We thus construct one scenario that generates the best case energy performance for both buildings considered jointly, as well as a second scenario that analogously results in the worst-base energy performance.

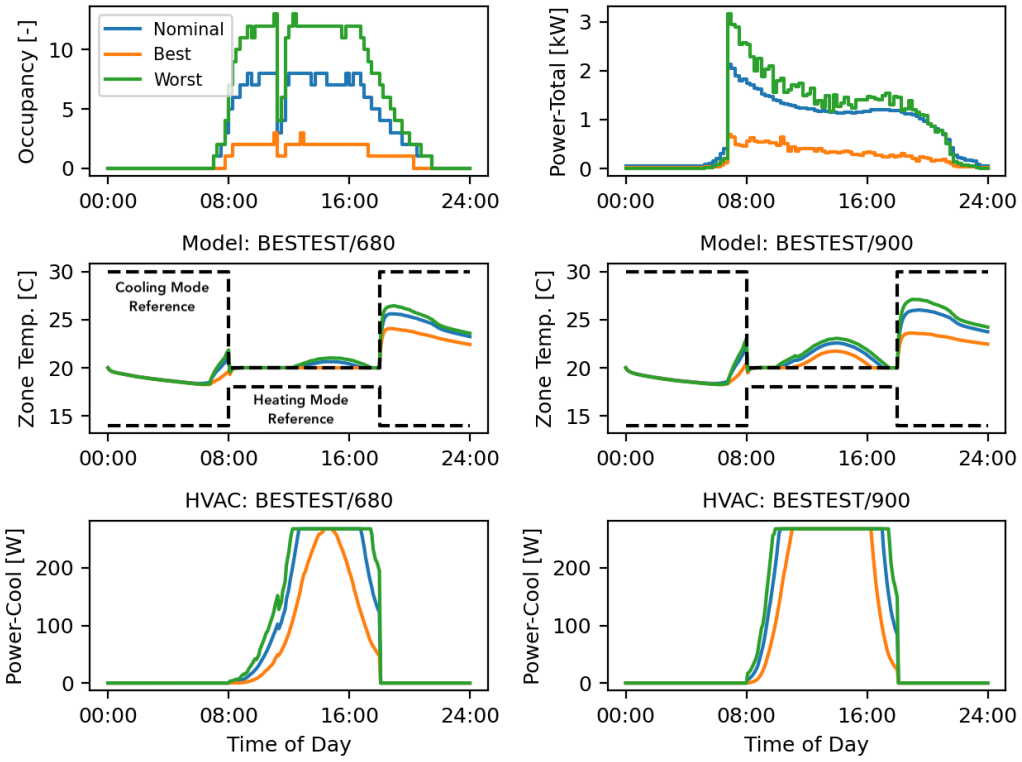

Figure 11: Demonstration of automatic deduction of limiting scenarios (occupancy and total internal energy use in the zone) by InfoBAX sampling of the latent space for two building constructions: BESTEST 680 (left column) which has low thermal mass and higher air infiltration, and BESTEST 900 (right column) which has high thermal mass and lower air infiltration. The corresponding zone temperatures and heat pump cooling profile is shown. Blue denotes a nominal scenario, and green/orange denotes the InfoBAX-inferred worst/best-case scenarios. The occupancy and total power has been generated by the RAFT-VG network, while the others are produced by simulation using these generated limiting scenarios.

We propose the use of InfoBAX [32] to efficiently identify these limiting scenarios by sequentially interacting with the building simulator. InfoBAX is a Bayesian experimental design algorithm that aims to infer outcomes of interest, denoted by $O_{\mathcal{A}}$, from an algorithm $\mathcal{A}$. We can think of an algorithm as performing a sequence of steps by interacting with a performance function $g : \mathcal{Z} \to \mathbb{R}$ that maps a value in the latent space $z \in \mathcal{Z}$ to a scalar performance value $g_z = g(z)$. Here, g_z represents the cumulative constraint violations throughout the simulated day for a given latent realizations z. The algorithm $\mathcal{A}$ of interest identifies the best- and worst-case elements from $\mathcal{Z}$ (i.e., the z values that produce the maximum and minimum value for g_z over z). The steps of algorithm $\mathcal{A}$ are: (i) evaluate g_z for all $z \in \mathcal{Z}$, (ii) sort the results $\{g_z\}_{z\in\mathcal{Z}}$ in decreasing order, and (iii) return the first and last elements. However, since g is an expensive-to-evaluate function, our budget for the number of evaluations is likely much smaller than the number of elements in $\mathcal{Z}$ (which could even be potentially infinite). InfoBAX helps us overcome this challenge by building a Gaussian process (GP) surrogate model for the black-box function g. This surrogate is then combined with an "acquisition function" that provides a measure of the information content of a new sample point. In particular, it uses the mutual information (MI) between a new sample point g_z and the outcome of the algorithm $O_{\mathcal{A}}$ given all available data $\mathcal{D}$, i.e., $\alpha(z|\mathcal{D}) = \mathsf{MI}(g_z; O_{\mathcal{A}} \mid \mathcal{D})$. The MI can be expressed in terms of the Shannon entropy of the random variables $g_z|\mathcal{D}$ and $\mathcal{O}_A|\mathcal{D}$, which can be computed using the approximation procedure in [32, Section 3.3]. We then sequentially find the sample that maximizes this information measure $z^\star = \arg\max_{z\in\mathcal{Z}} \alpha(z|\mathcal{D})$, evaluate the performance function at this point $g_{z^\star} = g(z^\star)$, update the dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \{(z^\star, g_{z^\star})\}$, and repeat.

Figure 11 illustrates the two scenarios identified by InfoBAX on the cases 680 and 900 simulation models, given a budget of 300 closed-loop simulations across a set of 5000 latent values randomly generated within the range of $[-5, 5]$ for each latent dimension. The best-case and worst-case occupancies are illustrated in the top left subplot, and

the internal heat load for both of these buildings under both of these scenarios is provided in the top right subplot. The zone temperature profiles over a 24-hour period under PI regulation for cases 680 and 900 are shown in the middle subplot of the left-hand and right-hand columns, respectively, with the temperature allowed to float freely within the dashed black bands. These PI controllers are only activated if the zone temperature exceeds the upper limit or falls below the lower limit of the dashed lines. The activation of these PI controllers is evident in the from the bottom row of subplots, which shows the cooling power required to regulate the room temperature within specified constraints for cases 680 and 900 in the left-hand and right-hand columns, respectively. We also compare these scenarios against the nominal scenario, which represents the average of the true data.

The top row of subplots depict a subset of the disturbance inputs generated by InfoBAX, namely the occupancy and internal heat load for the best- and worst-case scenarios. Each subplot in the top row shows the bottom-1 (worst-case) scenario shown in green, the nominal scenario in blue, and the top-1 (best-case) scenarios in orange. In summary, the bottom-1 scenarios are characterized by high occupancy throughout the workday, with significant use of lighting and equipment, tapering off as the workday ends. Conversely, the top-1 scenarios feature low occupancy and minimal use of lights and equipment.

In considering the effect of these limiting scenarios on these two test cases via the zone temperatures and cooling power, these simulations reveal the counter-intuitive impact of reducing infiltration and improving insulation quality on building performance. Due to higher occupant and internal heat loads compared to the effect of cold weather on the envelope, case 680 requires cooling from the HVAC system. This system reaches its performance limit around midday, causing the room temperature to rise slightly above the constraint from approximately noon to 6 pm. In contrast, the reduced infiltration and improved insulation for case 900 mitigate much of the cooling provided by the low ambient temperatures, making the building prone to overheating. Consequently, the cooling coil reaches its maximum capacity limit much earlier in the day, preventing the system from managing the heat gains effectively, resulting in the room temperature rising well above the desired setpoint for a significant portion of the day. These undesired thermal behaviors caused by 'improving' the building envelope are commonly observed, highlighting the importance of properly sizing HVAC systems to avoid such issues.

6. Conclusions

Efforts to describe and design the dynamics of physical systems in a human-centric context are necessarily dependent on the models of occupant behavior that inform the temporal evolution of experimental systems. In a buildings context, such occupant behavior models are best learned from data that captures the observed sensible and latent heat loads related to metabolic processes, as well as loads related to human activity. Whereas existing approaches often use deterministic models of human behavior or stochastic models that are suited to relatively small problems, we have developed a scalable approach to learn occupant behavior models directly from real-world data for the purposes of generating synthetic data of stochastic disturbances for simulations of physical systems. Furthermore, we have prototyped a framework that provides a means of integrating the proposed occupant behavior models with existing simulation tools, and demonstrated the benefits of this framework in energy monitoring and control evaluations for building operations.

This integration of deep generative models with high-fidelity building simulation tools represents a significant advancement in the field of occupant-centric building design and control. By leveraging the RAFT-VG architecture, we generated synthetic scenarios that closely mirror real-world occupancy patterns and internal heat loads, providing a more accurate assessment of building performance. This approach offers a scalable solution for evaluating control systems under a wide range of conditions, thereby enhancing the robustness of building designs and reducing the risks associated with suboptimal performance in real-world applications.

This framework for integrating physics-based simulation tools with deep generative models has a variety of potential applications beyond building simulation. Dynamic power grid simulations represent one such opportunity, as the availability of measurement data representing the behavior of electrical power consumption at a substation would enable a time-series generative network model to be trained and used to drive the behavior of a load in a power system simulation model instead of using a probabilistic model [33]. Any exogenous input that is not captured by conventional grid "physics" could be a potential candidate to be modeled with this proposed model. Such capabilities could also be used in the analysis of demand response (DR) scenarios for estimating the potential of DR in a residential

setting [34] to describe the exogenous inputs of temperatures and solar radiation, as well as replace the Markov chain models used to represent occupancy and electrical load consumption.

Potential areas of future work include further improvements to the accuracy of the generative models by incorporating additional covariates, such as weather conditions and external disturbances, to better capture the full range of factors influencing building performance. Additionally, the development of more efficient sampling methods for identifying limiting scenarios, such as those that push the boundaries of comfort and energy efficiency, could provide valuable insights for optimizing HVAC systems. Finally, the integration of generative models with reinforcement learning techniques offers the potential for developing adaptive control systems that can dynamically adjust to changing occupancy patterns and environmental conditions, thereby further enhancing the energy efficiency and comfort of modern occupant-centric buildings.

References

[1] Y. Pan, M. Zhu, Y. Lv, Y. Yang, Y. Liang, R. Yin, Y. Yang, X. Jia, X. Wang, F. Zeng, et al., Building energy simulation and its application for building performance optimization: A review of methods, tools, and case studies, Advances in Applied Energy 10 (2023) 100135.

[2] C. Y. Siu, W. O'Brien, M. Touchie, M. Armstrong, A. Laouadi, A. Gaur, Z. Jandaghian, I. Macdonald, Evaluating thermal resilience of building designs using building performance simulation–a review of existing practices, Building and Environment 234 (2023) 110124.

[3] Energyplus™, version 00 (9 2017).
URL https://www.osti.gov//servlets/purl/1395882

[4] M. Wetter, W. Zuo, T. S. Nouidui, et al., Modelica buildings library, Journal of Building Performance Simulation 7 (4) (2014) 253–270.

[5] P. X. Zou, X. Xu, J. Sanjayan, J. Wang, Review of 10 years research on building energy performance gap: Life-cycle and stakeholder perspectives, Energy and Buildings 178 (2018) 165–181.

[6] W. Tian, Y. Heo, P. De Wilde, Z. Li, D. Yan, C. S. Park, X. Feng, G. Augenbroe, A review of uncertainty analysis in building energy assessment, Renewable and Sustainable Energy Reviews 93 (2018) 285–301.

[7] D. Yan, W. O'Brien, T. Hong, X. Feng, H. B. Gunay, F. Tahmasebi, A. Mahdavi, Occupant behavior modeling for building performance simulation: Current state and future challenges, Energy and buildings 107 (2015) 264–278.

[8] Y. Chen, T. Hong, X. Luo, An agent-based stochastic occupancy simulator, Building Simulation 11 (2018) 37–49.

[9] Z. Chen, C. Jiang, Building occupancy modeling using generative adversarial network, Energy and Buildings 174 (2018) 372–379.

[10] A. Salatiello, Y. Wang, G. Wichern, et al., Synthesizing building operation data with generative models: VAEs, GANs, or something in between?, in: Companion Proceedings of the 14th ACM International Conference on Future Energy Systems, 2023, pp. 125–133.

[11] D. P. Kingma, M. Welling, Auto-Encoding Variational Bayes, in: 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014. arXiv:http://arxiv.org/abs/1312.6114v10.

[12] K. Sohn, H. Lee, X. Yan, Learning structured output representation using deep conditional generative models, Advances in neural information processing systems 28 (2015).

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, et al., Generative adversarial nets, in: Advances in neural information processing systems, 2014, pp. 2672–2680.
URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[14] M. Mirza, S. Osindero, Conditional generative adversarial nets, 2014.
URL http://arxiv.org/abs/1411.1784

[15] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein generative adversarial networks, in: International conference on machine learning, PMLR, 2017, pp. 214–223.

[16] T. Miyato, T. Kataoka, M. Koyama, et al., Spectral normalization for generative adversarial networks, 2018.

[17] F. Codeca, F. Casella, Neural Network Library in Modelica, in: 5th International Modelica Conference, 2006, pp. 549–557.
URL https://modelica.org/events/modelica2006/Proceedings/sessions/Session5c3.pdf

[18] Modelica Association, Functional Mockup Interface for Model Exchange and Co-Simulation, Version 2.0.1 (2019).
URL www.fmi-standard.org

[19] The MathWorks, Export Network as FMU, accessed: Feb. 2, 2024. Available since v. 2023b.
URL https://www.mathworks.com/help/deeplearning/ug/export-network-to-fmu.html

[20] XRG Simulation GmbH, SmartInt: Simple modelica artificial intelligence interface, accessed: Feb. 2, 2024.
URL https://github.com/xrg-simulation/SMArtIInt

[21] Wolfram Research, NeuralNet: Library that provides support for the use of neural networks in modeling, accessed: Feb. 2, 2024.
URL https://reference.wolfram.com/system-modeler/libraries/SystemModelerExtras/SystemModelerExtras.NeuralNet.html

[22] G. Schweiger, C. Gomes, G. Engel, et al., An empirical survey on co-simulation: Promising standards, challenges and research needs, Simulation modelling practice and theory 95 (2019) 148–163.

[23] Modelica Association, Modelica Language Specification, Version 3.6., accessed: Feb. 2, 2024.
URL https://specification.modelica.org/maint/3.6/functions.html#S9.p2

[24] D. Bruck, H. Elmqvist, H. Olsson, et al., Dymola for Multi-Engineering Modeling and Simulation, in: 2nd International Modelica Conference, 2002, pp. 55–1 — 55–8.
URL https://modelica.org/events/Conference2002/papers/p07_Brueck.pdf

[25] Dassault Systemes AB, Dymola — Dynamic Modeling Laboratory — Full User Manual, Lund, Sweden (September 2023).

[26] F. Jorissen, M. Wetter, L. Helsen, Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation, in: Proceedings of the 11th International Modelica Conference, 2015, pp. 59–69.
URL http://dx.doi.org/10.3384/ecp1511859

[27] S. Dorado-Rojas, M. N. Catalan, M. de Castro Fernandes, et al., Performance benchmark of Modelica time-domain power system automated simulations using Python, in: American Modelica Conference, Boulder, CO, 2020, pp. 28–34.

[28] ASHRAE, 140: Standard method of test for the evaluation of building energy analysis computer program, Tech. rep., ASHRAE (2007).

[29] Z. Lin, A. Jain, C. Wang, et al., Using GANs for sharing networked time series data: Challenges, initial promise, and open questions, in: Proceedings of the ACM Internet Measurement Conference, 2020, pp. 464–483.

[30] ASHRAE American Society of Heating Refrigerating and Air-Conditioning Engineers, ASHRAE Handbook - Fundamentals, ASHRAE, Atlanta, GA, 2017.

[31] Y. Sun, L. Gu, C. J. Wu, G. Augenbroe, Exploring hvac system sizing under uncertainty, Energy and Buildings 81 (2014) 243–252.

[32] W. Neiswanger, K. A. Wang, S. Ermon, Bayesian algorithm execution: Estimating computable properties of black-box functions using mutual information, in: International Conference on Machine Learning, PMLR, 2021, pp. 8005–8015.

[33] B. Mukherjee, M. de Castro Fernandes, L. Vanfretti, A pmu-based control scheme for islanded operation and re-synchronization of der, Int. J. of Electrical Power and Energy Systems 133 (2021) 107217.

[34] B. J. Johnson, M. R. Starke, O. A. Abdelaziz, R. K. Jackson, L. M. Tolbert, A dynamic simulation tool for estimating demand response potential from residential loads, in: 2015 IEEE Power and Energy Society Innovative Smart Grid Technologies Conference (ISGT), IEEE, 2015, pp. 1–5.