USE OF DIGITAL TWINS TO MITIGATE COMMUNICATION FAILURES IN MICROGRIDS

Andrew Eggebeen
University of Wisconsin-Milwaukee

Follow this and additional works at: https://dc.uwm.edu/etd

Part of the Computer Sciences Commons, and the Electrical and Electronics Commons

Recommended Citation
Eggebeen, Andrew, "USE OF DIGITAL TWINS TO MITIGATE COMMUNICATION FAILURES IN MICROGRIDS" (2023). Theses and Dissertations. 3400.
https://dc.uwm.edu/etd/3400

This Dissertation is brought to you for free and open access by UWM Digital Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of UWM Digital Commons. For more information, please contact scholarlycommunicationteam-group@uwm.edu.
USE OF DIGITAL TWINS TO MITIGATE COMMUNICATION FAILURES IN MICROGRIDS

by

Andrew R. Eggebeen

A Dissertation Submitted in
Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
in Engineering

at
The University of Wisconsin-Milwaukee
December 2023
ABSTRACT

USE OF DIGITAL TWINS TO MITIGATE COMMUNICATION FAILURES IN MICROGRIDS

by

Andrew R. Eggebeen

The University of Wisconsin-Milwaukee, 2023
Under the Supervision of Professor Robert M. Cuzner

This work investigates digital twin (DT) applications for electric power system (EPS) resilience. A novel DT architecture is proposed consisting of a physical twin, a virtual twin, an intelligent agent, and data communications. Requirements for the virtual twin are identified. Guidelines are provided for generating, capturing, and storing data to train the intelligent agent. The relationship between the DT development process and an existing controller hardware-in-the-loop (CHIL) process is discussed. To demonstrate the proposed DT architecture and development process, a DT for a battery energy storage system (BESS) is created based on the simulation of an industrial nanogrid. The creation and validation of the BESS DT virtual twin and intelligent agent are emphasized, including a discussion of the design choices made during the process. The use of data communication for nanogrid coordination is introduced, including the possible detrimental effects of degraded or failed communication. The BESS DT is demonstrated during nominal and off-nominal events in the nanogrid, highlighting the DT’s ability to make decisions using only local measurements rather than relying on a data communication network for coordination. The results show that the BESS DT can increase nanogrid resilience by recommending actions in response to transient events in the nanogrid, even while the data communication network has degraded or failed.
"If I have seen further, it is by standing on the shoulders of giants"
Sir Isaac Newton
# TABLE OF CONTENTS

1 Introduction  
1.1 Electric power system resilience ................................. 2  
  1.1.1 Microgrids ........................................... 2  
  1.1.2 Nanogrids ........................................... 3  
  1.1.3 Resilience ........................................... 3  
1.2 Problem description ............................................. 4  
1.3 Proposed solution .............................................. 6  
1.4 Chapter list ................................................... 6  

2 Literature Review  
2.1 Communication in EPS ........................................... 9  
2.2 AI/ML in EPS ............................................... 11  
2.3 Digital twins for EPS .......................................... 13  
2.4 Research gaps ................................................. 17  

3 Digital Twin Architecture  
3.1 Layers .......................................................... 20  
  3.1.1 Physical twin ........................................... 20  
  3.1.2 Virtual twin ........................................... 22  
  3.1.3 Intelligent agent ........................................ 24
# Table of Contents

5.6 Data communications .................................................. 60

6 Results........................................................................... 61
  6.1 Training .................................................................. 62
    6.1.1 Training process .................................................. 62
    6.1.2 Mini-batches ....................................................... 63
    6.1.3 Use of a GPU ....................................................... 63
    6.1.4 Loss functions ..................................................... 63
  6.2 Evaluation ................................................................. 64
    6.2.1 Confusion matrices .............................................. 65
  6.3 BESS DT demonstration plots ..................................... 68
  6.4 Discussion ............................................................... 70

7 Future Work ..................................................................... 78
  7.1 Implementation ........................................................... 79
  7.2 Improvements and related applications ......................... 80

8 Summary .......................................................................... 83
  8.1 DT architecture .......................................................... 84
  8.2 DT agent development process ..................................... 84
  8.3 BESS DT ................................................................. 85
  8.4 Key contributions ....................................................... 85
  8.5 Conclusion ............................................................... 86

A Virtual Twin Model .......................................................... 87

B Intelligent Agent ML Model .............................................. 92
LIST OF FIGURES

1.1 Microgrid resilience curve .................................................. 4
1.2 IEEE Std 1547-2018 Cat I ride through requirements .................. 5

3.1 DT architecture ................................................................. 21
3.2 Virtual twin flowchart .......................................................... 25

4.1 Real-time simulator interfacing with controller hardware .............. 37
4.2 Microgrid protection design process ....................................... 42
4.3 Development process for DT intelligent agent ............................ 43
4.4 DT training data storage and use cases .................................. 44

5.1 BESS DT diagram ............................................................... 46
5.2 Nanogrid diagram ............................................................... 48
5.3 Nanogrid data generation process .......................................... 50
5.4 Virtual twin predictions ........................................................ 53
5.5 Virtual twin prediction errors ............................................... 54
5.6 Virtual twin measurements and error vectors ............................ 55

6.1 Reconstruction loss ............................................................. 65
6.2 Action recommendation loss .................................................. 66
6.3 KL divergence loss .............................................................. 67
6.4 Confusion matrix, one cycle post-event .................................... 68
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Physical twin examples</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Virtual twin examples</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>Technical readiness levels</td>
<td>36</td>
</tr>
<tr>
<td>5.1</td>
<td>Nanogrid component ratings</td>
<td>47</td>
</tr>
<tr>
<td>5.2</td>
<td>Simulation fault types</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>Simulation fault locations</td>
<td>49</td>
</tr>
<tr>
<td>5.4</td>
<td>BESS DT agent actions</td>
<td>56</td>
</tr>
<tr>
<td>6.1</td>
<td>Correct recommendations post-event</td>
<td>67</td>
</tr>
</tbody>
</table>
NOMENCLATURE

Abbreviations

A   Ampere
AC  Alternating current
ADAM Adaptive moment estimation
ADC Analog-to-digital converter
AI  Artificial intelligence
ANN Artificial neural network
ATS Automatic transfer switch
BESS Battery energy storage system
CA  Certificate authority
CHIL Controller hardware-in-the-loop
CNN Convolutional neural network
CPS Cyber-physical system
CRL Certificate revocation list
DAC Digital-to-analog converter
DC Direct current
DER Distributed energy resource
DG Distributed generator
DNN Deep neural network
DT Digital twin
DWT Discrete wavelet transform
EPS Electric power system
FPGA Field programmable gate array
GPU Graphics processing unit
HDL Hardware description language
HPC High performance computing
Hz Hertz
I/O Input/Output
IBR Inverter-based resource
IM Induction machine
KL Kullback-Liebler
KV Kilovolt
KVA Kilovolt-ampere
KW Kilowatt
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>Long short-term memory</td>
</tr>
<tr>
<td>LV</td>
<td>Low voltage</td>
</tr>
<tr>
<td>LVAC</td>
<td>Low-voltage alternating current</td>
</tr>
<tr>
<td>LVRT</td>
<td>Low-voltage ride-through</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-agent system</td>
</tr>
<tr>
<td>MBSE</td>
<td>Model-based systems engineering</td>
</tr>
<tr>
<td>ML</td>
<td>Machine learning</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean squared error</td>
</tr>
<tr>
<td>MV</td>
<td>Medium voltage</td>
</tr>
<tr>
<td>NARX</td>
<td>Nonlinear autoregressive network with exogenous inputs</td>
</tr>
<tr>
<td>p.u.</td>
<td>Per unit</td>
</tr>
<tr>
<td>PCE</td>
<td>Polynomial chaos expansion</td>
</tr>
<tr>
<td>PF</td>
<td>Power factor</td>
</tr>
<tr>
<td>PHIL</td>
<td>Power hardware-in-the-loop</td>
</tr>
<tr>
<td>PM</td>
<td>Prime mover</td>
</tr>
<tr>
<td>PPO</td>
<td>Proximal policy optimization</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision time protocol</td>
</tr>
<tr>
<td>PV</td>
<td>Photovoltaic</td>
</tr>
<tr>
<td>PVECU</td>
<td>Photovoltaic energy conversion unit</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of service</td>
</tr>
</tbody>
</table>
RNN  Recurrent neural network
SDN  Software defined networking
SHA  Secure hash algorithm
SOA  Service-oriented architecture
SoC  State of charge
SVM  Support vector machine
TPM  Trusted platform module
TPU  Tensor processing unit
TRL  Technical readiness level
TSN  Time sensitive networking
V    Volt
VPP  Virtual prototyping process
VSC  Voltage source converter
W    Watt
ACKNOWLEDGMENTS

Committee members:

Dr. Robert Cuzner, University of Wisconsin - Milwaukee  
Dr. Susan McRoy, University of Wisconsin - Milwaukee  
Dr. Douglas Van Bossuyt, Naval Postgraduate School  
Dr. Chanyeop Park, University of Wisconsin - Milwaukee  
Dr. Jun Zhang, University of Wisconsin - Milwaukee

Organizations:

Grid-connected Advanced Power Electronic Systems (GRAPES)  
United States Office of Naval Research (ONR)  
University of Wisconsin - Milwaukee High Performance Computing Service

Funding provided by:

National Science Foundation Award No. 1939124  
NAVSUP Fleet Logistics Center Grant No. N00244-22-2-0002

For their support in general:

Dr. Hossein Hosseini, University of Wisconsin - Milwaukee  
Everyone in the Center for Sustainable Electric Energy Systems lab

I owe my gratitude to far more people than those mentioned here. This work would not have been what it is without their patience and encouragement.
Chapter 1

Introduction
1.1 Electric power system resilience

Energy security is critical for both national security and quality of life. An electric power system (EPS) supports energy security by delivering electric power where it’s needed, when it’s needed. Loss of power to critical services and infrastructure, such as medical facilities or water treatment plants, would present a danger to health and well-being. An EPS must be able to maintain quality power service even during off-nominal events such as equipment failure or damage, electrical faults, communication faults, and cyberattacks.

Microgrids and nanogrids have been proposed as a means of increasing EPS resilience. Microgrids and nanogrids can decentralize control of the EPS by operating even while disconnected from a larger utility grid. The control and coordination methods in microgrids and nanogrids often rely on a communication network. Communication networks are inherently unreliable, therefore the microgrid and nanogrid control systems must have the ability to continue operating even while the communication network has failed.

1.1.1 Microgrids

Microgrids are smaller power grids with localized renewable energy resources, energy storage, and fuel-based generators along with several loads. Microgrids typically contain inverter-based resources (IBR) which utilize power electronics to convert energy. Microgrids can operate both connected to a utility grid or islanded (disconnected) from it. A microgrid is tailored specifically to its installation and to the installation’s unique set of requirements. Until recently, insufficient attention has been paid to the microgrid interactions with the installation’s distribution network. This leads to degradation of the microgrid’s resilience while operating in islanded mode. An overview of resilience is provided in Section 1.1.3.

The use of renewable energy resources in microgrids promotes energy security [1].
The continued use of fuel requires continual replenishment, and a disruption to the fuel supply could disrupt the availability of electric power. Solar energy, wind energy, and energy from other natural phenomenon can be harnessed to reduce the demand for fuel. Generators may be shut off while there is a sufficient supply of energy from renewable resources or energy storage. Running a generator less frequently may also increase its operating life and lower its maintenance costs. The cost of deploying renewable resources may be offset by the savings from reduced fuel and maintenance costs, thus adding a potential economic benefit to the use of microgrids as well.

1.1.2 Nanogrids

Nanogrids are fundamental topologies for building a larger EPS, including a microgrid. Similar to microgrids, nanogrids can operate in islanded mode. Nanogrids can be used to create zones of protection in an EPS [2, 3], and by minimizing nanogrid topologies, faults can be isolated to minimal areas of the EPS.

During islanded operation, the distributed energy resources (DER), nanogrid controllers, and switchgear/automatic transfer switches (ATS) must coordinate to properly and efficiently deliver power. This coordination typically takes place through an exchange of command and control messages over a data communication network. Communication may also take place both within a nanogrid itself and with other parts of the larger EPS.

1.1.3 Resilience

An IEEE task force on "Resilience Framework, Methods, and Metrics for the Electricity Sector" defines resilience as "The ability to protect against and recover from any event that would significantly impact the grid." [4]. Another IEEE task force on "The Definition and Quantification of Resilience" defines resilience as "The ability to withstand and reduce the
Figure 1.1: Microgrid resilience curve

magnitude and/or duration of disruptive events, which includes the capability to anticipate, absorb, adapt to, and/or rapidly recover from such an event.” [5]. In general, resilience is the ability to mitigate off-nominal events occurring in an EPS.

The resilience of an EPS includes its ability to respond, adapt, and recover during off-nominal events. Attributes of resilience can be measured with resilience metrics and various mathematical methods [6,7]. Fig. 1.1 provides a qualitative curve which can be used to assess the state of a microgrid in terms of resilience [8].

1.2 Problem description

Using a communication network for coordination in a microgrid or nanogrid requires that the impact of nominal and off-nominal communication conditions must be assessed under both nominal and off-nominal electrical conditions. Simultaneous communication and electrical faults are an extreme corner case for which solutions must be designed.
IEEE Std 1547-2018 [9] provides guidance for connecting DERs in distribution systems. Proper coordination between the distribution equipment protection settings and the DER low-voltage ride-through (LVRT) settings in the power conversion equipment is needed to ensure the maximum number of DERs will ride through a fault [8].

If IEEE Std 1547-2018 Category I ride through requirements (Fig. 1.2) are applied to a battery energy storage system (BESS) in a nanogrid, the BESS converter must know under which conditions it should disconnect, perform momentary cessation, or switch to grid-forming mode. These actions are often coordinated through a communication network. If the communication network is inoperable during an electrical fault, the BESS will be unable to receive commands or query other devices for information regarding the fault.

To maximize resilience, the BESS needs to take action even while the communication network has degraded or failed. This requires that the BESS be capable of making decisions and taking action autonomously. Since the actions taken by the BESS will affect the overall operation of the nanogrid, the BESS must estimate which action will best support resilient operation. Making appropriate decisions in the case of simultaneous communication and electrical faults requires a method which can analyze fault characteristics using only local observations and measurements.

<table>
<thead>
<tr>
<th>Voltage range (p.u.)</th>
<th>Operating mode/response</th>
<th>Minimum ride-through time (s) (design criteria)</th>
<th>Maximum response time (s) (design criteria)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V &gt; 1.20$</td>
<td>Cease to Energize</td>
<td>N/A</td>
<td>0.16</td>
</tr>
<tr>
<td>$1.175 &lt; V \leq 1.20$</td>
<td>Permissive Operation</td>
<td>0.2</td>
<td>N/A</td>
</tr>
<tr>
<td>$1.15 &lt; V \leq 1.175$</td>
<td>Permissive Operation</td>
<td>0.5</td>
<td>N/A</td>
</tr>
<tr>
<td>$1.10 &lt; V \leq 1.15$</td>
<td>Permissive Operation</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>$0.88 \leq V \leq 1.10$</td>
<td>Continuous Operation</td>
<td>Infinite</td>
<td>N/A</td>
</tr>
<tr>
<td>$0.70 \leq V &lt; 0.88$</td>
<td>Mandatory Operation</td>
<td>Linear slope of 4 s/1 p.u. voltage starting at 0.7 s @ 0.7 p.u.: $T_{VRT} = 0.7 s + \frac{4}{1}V(0.7 - 0.7 \ p.u.)$</td>
<td>N/A</td>
</tr>
<tr>
<td>$0.50 \leq V &lt; 0.70$</td>
<td>Permissive Operation</td>
<td>0.16</td>
<td>N/A</td>
</tr>
<tr>
<td>$V &lt; 0.50$</td>
<td>Cease to Energize</td>
<td>N/A</td>
<td>0.16</td>
</tr>
</tbody>
</table>

Figure 1.2: IEEE Std 1547-2018 Cat I ride through requirements
1.3 Proposed solution

While a precise definition of a digital twin (DT) may not yet be agreed upon, many proposed applications of DT employ the use of a digital model to analyze the behavior of a physical device or system in real-time.

DT have been created for entire systems and for individual devices. System-level DT rely on a communication network to gather measurements and send coordination commands. Device-level DT acquire their measurements locally and can make decisions autonomously. The digital model for a system-level DT may exist remotely, perhaps in a cloud-based service, while the digital model for a device-level digital twin is executed locally on a processor closely connected to the physical device.

Since device-level DT can operate locally and autonomously, their use can mitigate adverse affects caused by the degradation or failure of a communication network. They can operate during simultaneous communication and electrical faults by using local measurements to make intelligent decisions without the need for remote information.

This work proposes the use of DT as a means to recommend actions in real-time during nominal and off-nominal events, and to do so while the communication network has degraded or failed. To demonstrate the creation and use of a DT for this purpose, a DT is developed for a BESS attached to an industrial nanogrid, and the DT’s ability to recommend actions using only local measurements is assessed. The results of the work are discussed in Section 6.4.

1.4 Chapter list

The remainder of this work is arranged as follows:

- Chapter 2 contains a literature review of topics related DT and their use in EPS
- Chapter 3 presents the DT architecture and provides details on each of its layers
• Chapter 4 presents a development process for DT and explains the relationship of the process to existing hardware development processes

• Chapter 5 explains the development and operation of the BESS DT

• Chapter 6 describes the training process for the BESS DT and contains the results

• Chapter 7 provides a discussion on future work

• Chapter 8 provides a summary and a list of key contributions from this work

• Appendix A contains details regarding the BESS DT virtual twin

• Appendix B contain details on the BESS DT intelligent agent
Chapter 2

Literature Review
2.1 Communication in EPS

Cyber-physical systems (CPS) may be characterized as "a set of physical devices, objects and equipment that interact with a virtual cyberspace through a communication network" [10], "a system that integrates equipment, sensors, computing resources, and information systems throughout the entire value chain" [11], and "seamlessly interweaving the physical world of infrastructure objects and the virtual world of information processing" [12].

The U.S. Department of Energy "Grid 2030" [13] is the vision of a fully automated power delivery network with "distributed intelligence, coupled with broadband communications and automated control systems" as key enabling technologies. Physical assets of a power grid (generators, energy storage systems, power electronic converters, protective equipment, etc.) managing and coordinating their operation over a communication network means this smart power grid can be viewed as a cyber-physical system [14–22].

Microgrids and nanogrids share many similar concepts with this view of the smart grid, including the use of a communication network as a core part of their operation. An overview of communication technologies in microgrids and smart grids can be found in [23–27]. A management or coordination scheme which leverages the benefits of a communication network must also take into account the undesirable effects of networks such as latency, jitter, lost or discarded data, and the possibility of security issues including equipment malfunctions and cyberattacks. Several works have demonstrated detrimental effects on EPS caused by off-nominal communications [28–37].

Use of a communication network to coordinate fault recovery is demonstrated in [38]. Devices in the microgrid continually exchange their self-reported status with their neighbors. The exchange of information such as power output can be utilized to detect electrical faults, while communication faults can be detected by excessive delayed or missing status updates from neighbors. When a problem is detected, the controllers can follow their prescribed fault recovery strategy.

A method for handling communication delays in DC microgrids is proposed in [34].
DER controllers in the system exchange messages to meet the power sharing objectives. The controllers also calculate the maximum amount of time between these messages before stability would be lost, and messages are sent proactively to prevent loss of stability due to communication delays.

In [35], a model for assessing the impact of communication delays on DC microgrid stability is presented. A secondary control algorithm is formulated and stability criterion are divided into delay-dependent and delay-independent categories. Controller parameters can be set based on these stability criterion, and the system can maintain stability up to the maximum allowable communication delay.

An approach to determining the requirements of a communication network is given in [39]. A model of a DC microgrid is formulated to examine the system response considering the communication network topology alongside the microgrid physical specifications (load ratings, transmission impedance, etc.). The proposed model can used to study several communication topologies and how they impact system performance under different microgrid specifications. By understanding the interaction of the communication network and the microgrid power network, critical parts of the communication network can be identified.

Communicating devices must have a shared understanding of the information being exchanged, and ontologies can be used to represent this knowledge. Ontologies are used “to create a formal representation of real-world systems, the objects composing that system, and the relationships between those objects” [40]. Sharing an ontology between devices facilitates interoperability, since all devices understand the exchanged data in the same manner. Use of ontologies has been shown to be a key part of CPS [41–43], and existing microgrid communication methods including IEC 61850 and CIM can be mapped to ontologies [44].
2.2 AI/ML in EPS

As the complexity of an EPS increases, it becomes more difficult to create a mathematical model of it. Nonlinear effects from IBR and other power electronics will further complicate this difficulty. Techniques from artificial intelligence (AI) and machine learning (ML) have been used to address this complexity, and the use of AI/ML can create more decentralized and resilient control of an EPS.

A protection scheme for microgrids is proposed in [45]. Features from voltage and current measurements are extracted, such as RMS, standard-deviation, energy, and Shannon-entropy. These features are used as input to a support vector machine (SVM). The SVM classifies these features as either fault or no-fault. After a fault is detected, distributed generators (DG) inject distinct harmonics onto the bus, and these harmonics are used to determine which breakers should trip to isolate the fault.

A method for local protection is proposed in [46]. The IEEE 123-node system is split into zones connected through relays. SVM classifiers are used to classify faults, estimate EPS topology, and detect which zone of the EPS is faulted. Voltage and current measurement features are used as inputs to the SVMs, including active/reactive power, RMS values, and zero-sequence components. The SVMs are capable of accurately classifying faults, even with different EPS topologies.

A artificial neural network (ANN) approach to fault classification in a microgrid is given in [47]. A test microgrid consisting of four inverter-based DGs is created. The output current of the DG are used as input to the ANN, and the ANN outputs a binary code to classify the currents. The ANN is trained to classify both line faults and induction motor faults.

Faults in a DC microgrid are detected and classified in [48] with a recurrent neural network (RNN). Different events are simulated, including faults and load/generation changes. Instantaneous bus voltage and current measurements are used as input to the RNN, and the RNN outputs which event has occurred. The authors of [48] demonstrate
the trade-offs on classification accuracy while changing the number of classification labels (e.g. by classifying similar faults into a single class).

A long short-term memory (LSTM) network is used in [49] to detect microgrid islanding. Voltage and current measurements are taken at the circuit breaker (CB) between the microgrid and the utility grid. Frequency components of these measurements are extracted and passed to the LSTM as input, and the LSTM outputs whether the microgrid is operating in either islanded mode or non-islanded mode.

A convolutional neural network (CNN) is used to detect and locate faults in [50]. Voltage measurements from all busses in the IEEE 9-bus and 39-bus models are used as inputs to the CNN. Faults are induced on each bus, and the CNN outputs which bus (if any) is faulted.

Both CNN and LSTM were used for fault classification in [51]. Voltage measurements from each phase of a modified IEEE 13-bus model are filtered with a wavelet transform, then passed as input to each of the networks. The networks output which type of fault is occurring. The authors of [51] also provide a comparison the CNN and LSTM networks and found that both were able to achieve a high degree of fault classification accuracy with the LSTM network requiring less data to do so.

Faults were detected and classified with a LSTM in [52]. A microgrid model based on the IEEE 13-node system was used. Instantaneous voltage and current measurements from each phase are used as input, and the LSTM output which fault (if any) is occurring. No data preprocessing was used, and the authors of [52] found the LSTM could accurately classify faults within 1/8 of a 60 Hz cycle.

An ANN was used in [53] to alleviate the effect of cascading failures due to breakers tripping in a microgrid. Breaker trip signals for each transmission line are used as inputs to the ANN. The ANN outputs power dispatch signals for each generator in the microgrid, causing them to change power output in response to the tripped breakers. The authors of [53] compared fault scenarios both with and without the ANN and found that
use of the ANN increased the stability of the system (e.g. bus voltage and frequency) following breakers tripping.

Faults are detected and located in a DC microgrid using an ANN in [54]. Current measurements from relays in the microgrid are decomposed with a discrete wavelet transform (DWT). The wavelet coefficients are used as inputs to the ANN, and the ANN outputs which fault is occurring. The authors of [54] note the dynamic nature of events in a microgrid along with the importance of capturing temporal dependencies of the measurements in the ANN.

A reinforcement learning approach to energy management is provided in [55]. An intelligent agent is created to operate in a microgrid containing a single load, a photovoltaic (PV) source, a battery, and a desalination unit. The agent measures data including the PV power production, the battery state of charge (SoC), the desalination water level, and the power demand from the load. The agent can take actions including charging or discharging the battery and operating or not operating the desalination unit. Combinations of measurement levels are encoded to form several states, and the agent can perform actions to attempt to move the system to another state. A Q-learning approach is used to train the agent on which actions it should take, and after training the agent it is able to maintain a good balance between battery SoC and operation of the desalination unit. A similar discussion on reinforcement learning is provided in [56].

2.3 Digital twins for EPS

Early definitions of DT can be found in [57,58]. A DT is described in [58] as “an integrated multiphysics, multiscale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin”. This idea of a DT was proposed for monitoring the overall health of NASA aircraft, but it’s fundamental concepts are applicable to other disciplines which involve
A DT for smart grids is presented in [59]. ANGEL monitors a smart grid using SCADA and other communication techniques to gather measurements in real-time. These measurements are used by a simulation of the grid, which is also run in real-time. By comparing the received measurements to the measurements predicted by the DT simulation, anomalies including electrical faults and false data injection can be detected. The authors of [59] note the need for real-time communication, thus identifying communication delays or failures as a potential weakness.

ANNs are employed in [60] to locate faults in a smart grid. Measurements from smart meters in the grid are sent across a communication network and used as inputs to the ANN. The ANN outputs are used to detect and locate faults. Use of the ANN aids in the analysis of a fault by simplifying the transient state estimation (TSE) equation, allowing the TSE to be solved only for the faulted parts of the system rather than the entire system. The authors of [60] also note the need for real-time communication.

Use of a deep learning CNN for smart grid anomaly detection is presented in [50]. The CNN is demonstrated on the IEEE 9-bus model and the IEEE 39-bus model. Three-phase voltages from each bus are used as inputs to the CNN. The CNN detects faults and estimates which bus in the model has been faulted.

A DT for a battery and its driver is presented in [16]. A simulation of the battery and driver was created, and a rule-based system was used to detect anomalies in the battery state of charge. Anomalies were detect by a divergence between the predicted SoC and actual SoC. The authors of [16] identified the importance of an accurate model of the system and noted the need for timely communication between the battery system and the simulation. A related work [17] improves the battery model by training a CNN to predict the battery system response.

A medium voltage (MV) to low voltage (LV) distribution transformer DT is given in [61]. Voltage and current measurements are taken on the LV side of the transformer.
A simplified model of the transformer circuit is used to estimate the voltages and currents on the MV side of the transformer given the LV side measurements. The authors of [61] use a statistical comparison and a frequency analysis of the model output, testing its predictions against data recorded from a real transformer.

In [62], a nonlinear autoregressive network with exogenous inputs (NARX) is used to model the dynamics of a power converter. The NARX model is used to predict inductor current and capacitor voltage for the converter given the converter’s duty cycle, input voltage, and load current. Once trained, the NARX model can be run in real-time alongside the converter to predict the expected converter measurements. The model is validated by testing its response in both the time domain and the frequency domain.

A DT is recommended for self-security of inverters in [63]. The DT contains knowledge of IEEE 1547 requirements and of the safe operating region and dynamic response of the inverter. Commands to change the PQ setpoints can be sent to the inverter, and the DT will compare these setpoints to its knowledge of the inverter and the current state of the system. If the PQ setpoints would cause a problem, the DT rejects the setpoints. Rejecting invalid PQ setpoints keeps the inverter in a safe state, protecting against both malicious and unintentionally dangerous commands.

A DT for a photovoltaic energy conversion unit (PVECU) is demonstrated in [64]. The DT contains a state-space model of the converter to model the inductor current and output capacitor voltage. The DT also contains knowledge of the maximum power point (MPP) current and voltage of the PV panel. Using measurements of the inductor current, capacitor voltage, PV irradiance levels, and PV panel temperature, the DT predicts what the measurements will be at the next timestep. An error vector at a given timestep is calculated as the difference between the predicted measurements and the actual measurements at that timestep. Tolerance levels are set for each dimension of the error vector, and the DT signals a fault if any of the tolerances are exceeded. An inner-product calculation is used to compare the trajectory of the error vector to a library of pre-calculated
fault vectors, using the result to predict which type of fault (if any) is occurring.

A probabilistic approach to analyzing error vectors is given in [65]. The state-space model is formulated where the coefficient matrices are treated as stochastic processes. The error vector is calculated similar to [64], but the use of polynomial chaos expansion (PCE) allows the tolerance levels of the error vector to be varied probabilistically. This approach can model stochastic effects of the physical device, such as circuit components being manufactured within a tolerance of their rated value. This allows the DT to detect faults while permitting the expected variances in its physical counterpart.

DT are being identified as a core part of CPS [41, 66, 67]. The physical assets can be modelled in virtual space, where "each physical entity has its cyber part as its digital representation, culminating in the DT" [41]. The summary of DT applications in cyber-physical systems presented in [68] cites the use of DT for reasons including:

- Monitoring for anomalies or fatigue in physical assets
- Modelling the reliability of a physical system
- Studying long-term behavior of a system including its interaction with its environment
- Optimizing performance based past and present states.

More than just a simulation of its physical counterpart, a DT is "a living, intelligent and evolving model, being the virtual counterpart of a physical entity or process" [41]. Several levels of DT sophistication are provided in [69], with the most sophisticated DT being an "Intelligent Digital Twin". An intelligent DT is capable of learning about its physical twin and the surrounding environment, which enables the DT to operate with a high degree of autonomy.

An autonomous system has "the ability to respond to unexpected events in an intelligent and efficient manner without the need for reconfiguration at the supervisory level" [70].
Self-awareness [71] may give an autonomous system the ability to “explicitly reason and learn about its own components and capabilities”. Four main areas of self-management are discussed in [72]: self-configuration, self-optimization, self-healing, and self-protection. Each of these self-management capabilities, which can be facilitated by DTs, can play a part in maximizing EPS resilience.

Several overviews of DTs and their use cases in CPS and EPS can be found in [41, 66–69, 73–85]. In summary, DT are a means of creating decentralized, autonomous, and secure control of CPS including EPS. As such, the use of DT presents a viable opportunity to increase the resilience of microgrids and nanogrids.

2.4 Research gaps

Despite DTs being an active area of research, no agreed upon definition of a DT seems to exist. Many uses cases show what a DT does but not what a DT is. The idea of physical and virtual twins exists in nearly all DT work, but a clear concept of a virtual twin is lacking. Many works also include the use of a communication network and of artificial intelligence and machine learning (AI/ML), but few identify these as a standard part of a DT itself.

Many of the works in the literature review assume that the communication network is fully functional and provide no guidance or investigation into off-nominal communications. Several works identify off-nominal communications as a potential point of failure [16, 59, 60, 73]. The methods for handling communications delays [34, 35] have an upper bound on length of the delays. An approach to handling communication failures is proposed in [46], but this work is not specific to DTs. More research should be performed on DT operating during communication failures.

An example of an intelligent agent performing actions is provided in [55], but this work describes an intelligent agent in general, not for a DT specifically. Much work exists
on DTs being use to detect and identify problems (e.g. faults in an EPS), but work on DTs taking preventative or corrective actions is lacking.
Chapter 3

Digital Twin Architecture
3.1 Layers

Based on the use cases and expectations of DTs from the literature review, a four-layer DT architecture is proposed. Fig. 3.1 contains a diagram of these layers. The four layers of the DT are:

- Physical twin - the physical device or system itself
- Virtual twin - a real-time, physics-based digital replica of the physical twin
- Intelligent agent - the learning and decision making processes of the DT
- Data communications - an exchange of information with other devices, services, or human operators

3.1.1 Physical twin

The physical twin layer interfaces with its corresponding physical device or system. An example of a device-level physical twin may be the power converter for an energy storage system, while an example of a system-level physical twin may be the aggregate of all assets in a microgrid. Table 3.1 contains the physical twins as identified in the literature review.

The physical twin is equipped with sensors to take physical measurements such as voltage, current, temperature, magnetic flux, etc. The physical twin may also perform actions to assert a physical effect on the device or system or to modify its interaction with its environment. These actions could include operating digital outputs or changing controller parameters.

Measurements from the physical twin are passed to the virtual twin in real-time. Likewise, the virtual twin may send commands to control the physical twin [87]. The interaction between the physical twin and the virtual twin creates the cyber-physical integration
of the DT [11, 18, 21], allowing the DT to serve as a bridge between the real world and a digital system of interacting devices.

The method for exchanging data between the physical twin and the virtual twin depends on the application. In applications such as [50, 59, 60, 73], physical twin measurements are sent to the virtual twin over a communication network, while [64, 65] implement the virtual twin on an FPGA alongside the physical twin.

Since the real-time exchange of data between the physical and virtual twins is imperative, consideration must be given to any problems which may complicate this exchange,
such as latency or loss of packets in a communication network. A cyberattack may aim to disrupt or interfere with this communication [59]. The processing rate of the virtual twin may be limited by the rate at which it can receive data from the physical twin [31].

### 3.1.2 Virtual twin

The virtual twin is a real-time, up-to-date model of the physical twin. Since the physical twin is a real-world, tangible asset which will be providing physical measurements, its virtual twin model must be physics-based. To stay synchronized with the physical twin, the virtual twin model must be executed in real-time while physical twin measurements are being received. The virtual twin is more than simply "a real-time simulation" of the physical twin; it is expected to continually update its model using live measurements from the physical twin and to estimate future states of the physical twin. These expectations impose the following virtual twin requirements:

- **Physics-based**
- **Real-time capable**
- **Measures and predicts physical twin state**

The virtual twin model should be a sufficiently accurate representation of the physical twin [62, 70, 87]. However, the model of the physical twin may be either partially or
Table 3.2: Virtual twin examples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Virtual twin model</th>
</tr>
</thead>
<tbody>
<tr>
<td>[59]</td>
<td>System simulation</td>
</tr>
<tr>
<td>[60]</td>
<td>ANN</td>
</tr>
<tr>
<td>[50]</td>
<td>CNN</td>
</tr>
<tr>
<td>[16]</td>
<td>System simulation</td>
</tr>
<tr>
<td>[17]</td>
<td>System simulation</td>
</tr>
<tr>
<td>[61]</td>
<td>Differential equations</td>
</tr>
<tr>
<td>[62]</td>
<td>NARX</td>
</tr>
<tr>
<td>[64]</td>
<td>State-space model</td>
</tr>
<tr>
<td>[65]</td>
<td>Stochastic state-space model</td>
</tr>
<tr>
<td>[86]</td>
<td>Differential equations with adaptable parameters</td>
</tr>
</tbody>
</table>

completely unknown. The physical twin model can be viewed as a black-box (completely unknown), gray-box (partially known), or white-box (fully known) [86, 88].

In the absence of a fully known physical model, machine learning and optimization techniques may be used to create an approximate virtual twin model based on data gathered from its physical counterpart [62, 82, 86]. A hybrid modelling approach is presented in [86], using a gray-box model of the converter along with data measured from the converter to best fit the model parameters. Neural networks are used in [50, 60, 62] to learn mappings of the physical twin measurements to system behavior. The PCE approach in [65] allows the model to handle uncertainty in component values. A discussion of uncertainty modelling is given in [76]. Table 3.2 contains the virtual twins models as identified in the literature review.

The virtual twin’s understanding of the physical model can be used to monitor and protect the physical twin. Using its understanding of the physical twin dynamics, a virtual twin can monitor the health of the physical twin [59, 60, 64, 65], analyze its behavior and interaction with its environment [16, 17, 64, 65], and prevent it from being placed in a dangerous state [63]. The virtual twin also interfaces with the intelligent agent layer, allowing the agent to analyze the physical twin behavior and to recommend actions to be taken by the physical twin.
Since the DT will be interacting with the physical world through the physical twin, physical safety becomes a concern [20,31,63,89–92]. This creates safety expectations for the DT, for example "ensuring control-theoretic properties such as stability under attacks" [89]. An approach to DT safety is presented in [63] where changes to active and reactive power setpoints (PQ setpoints) are double-checked against the virtual twin model of the inverter. If the PQ setpoints would endanger the inverter, the setpoints are rejected, thus adding a means of physical safety to the DT.

An error vector can be calculated as the difference between the measurements predicted by a virtual twin and the measurements actually received from the physical twin [62,64,65,86]. Assuming the virtual twin model is sufficiently accurate, this error vector captures effects acting on the physical twin which are unobservable or not included in the virtual twin model. For example, [64] and [65] use the behavior of the error vector to detect and classify faults. Error vectors may be passed to the intelligent agent to analyze the operation of the physical twin. A flowchart of the measurement, prediction, and error vector calculation process is shown in Fig. 3.2

The ability to predict future states of the physical twin allows the DT to make an estimate of the physical twin state even if the real-time exchange of data between the physical and virtual twins is disrupted. As stated in [83], "mirroring the power system by a dynamic simulation engine has the advantage, that even in the case of non-availability of actual measurement data due to measurement failure or communication loss, a statement about the latest system state is still possible". This capability is critical when dealing with the inherent unreliability of communication networks.

### 3.1.3 Intelligent agent

The intelligent agent (or simply, "the agent") is a model of the learning and decision-making processes of the DT. The agent can use information from the virtual twin to reason about the physical twin, its dynamics, and its environment. The agent may also consider
many other information, such as goals issued by an operator or data from a knowledge base. The agent may interact with the physical twin by sending commands to the virtual twin. Agent behavior can be adaptable, and improving agent behavior allows the DT to more effectively interact with its environment and achieve its goals. IEEE provides recommendations for integrating intelligent agents with low-level automation in [93].

Intelligent agents interact with their environment through a sense-decide-act loop [94]. This loop has been identified in [70,95,96] and others as a core part of CPS operation. The agent receives information about its environment through sensors. After sensing the environment, the agent makes a decision about which action to take (if any). An action is performed by asserting an effect on the environment through actuators. For the DT, the steps of the sense-decide-act loop are: receive the physical twin sensor measurements, analyze the measurements and any other relevant data to make a decision on which ac-
tion to take, then perform the action by sending a command to control the physical twin actuators.

ML algorithms can be "applied to a data stream to uncover/discover patterns that can be subsequently exploited in a variety of ways" [69]. Use of ML can provide digital twins the ability to improve their operation [17, 50, 60, 62, 86], and ML can be used by an intelligent agent to facilitate complex and adaptable decision making. Some works formalize the self-adaptable decision making with a process known as MAPE-K: Monitor-Analyze-Plan-Execute over a Knowledge base [16, 71, 97, 98].

The DT agent may adapt to its environment over time based on a history its measurements and observations. The DT may periodically re-train its agent using these measurements and observations to improve its operations specific to its real-world deployment [99]. This training may take place locally or may take place through a remote or cloud-based service [12, 100, 101]. Use of a remote or cloud-based service may facilitate the incorporation of global knowledge into the training process of localized agents.

The "collect-organize pattern" is proposed in [98] for coordinating distributed systems. The collect-organize pattern is applicable to a distributed system which "provides services to autonomous, cyber-physical entities, whereby each entity generates their own, local models and collects data that is spatially distributed and continuously changing". The DT architecture matches this description of a cyber-physical entity. The pattern describes a system where local models of physical resources are updated with data from the resource and where entities in the system exchange their local models to collectively create a global model. By distributing global knowledge to local entities, local operation can be maintained even when global communication has failed.

In addition to facilitating the exchange of communicated data, an ontology allows an intelligent agent to formally reason about data [10, 40, 44, 102, 103]. The use of ontologies can increase the interoperability of systems [40, 104, 105], and ontologies can be used for security assessment of a system [106, 107]. As noted in [68], "the DT must therefore be
supported by a proper data model structuring information about the system operations, its history, its behaviour and its current state”. A DT may use an ontology to describe this data model, and a DT agent could use ontologies to make intelligent decisions when coordinating with the other DTs.

3.1.4 Communications

The communication layer allows the DT to exchange data with other devices, services, or human operators. The communication layer is responsible for the security of the exchanged data. Data security includes encryption (keeping data private), authentication (ensuring the origin of data), and integrity (ensuring data has not been altered). Communication networks are subject to intermittent operation and failure due to malfunctions, equipment damage, cyberattacks, etc. A DT must be capable of operating safely and autonomously while communications have degraded or failed.

Several communication standards for microgrids have borrowed from existing standards for electric power substations. For example, IEEE Std 1615 [108] provides guidance for implementing communication networks in electric power substations, and IEC 61850 defines data models, exchanges, and events for substation automation [109].

Quality of service (QoS) metrics are often used to measure the performance of communication networks. IEEE Std 1646 [110] provides QoS requirements specific to electric power substation automation. Some popular QoS metrics are:

- Throughput - the amount of data which can be transferred over a given time period
- Latency - the delay between data being sent and being received
- Jitter - the amount of variance in latency
- Packet loss - the amount of data which fails to be properly received, usually measured as a percentage of sent data
The real-time constraints of DT impose QoS requirements on their communication network [111]. Failure to meet these QoS requirements can have adverse effects [29–33]. Messages being sent across a communication network can be placed into three QoS categories [112]:

- Hard real-time - strict latency requirements, minimal packet loss
- Soft real-time - relaxed latency and jitter requirements, some packet loss acceptable
- Not time-critical - tolerates packet loss, minimal latency and jitter requirements

Several networking technologies aim to ensure QoS metrics will be met. Time-sensitive networking (TSN) is a networking technology being directed by an IEEE 802.1 task group. TSN can prioritize and control network traffic to guarantee certain QoS metrics are met. The tradeoff for this QoS guarantee is a limited network size. TSN has been shown to be useful for meeting the QoS requirements of messages used by CPS and DT [87, 112–114].

Communication problems can be detected by implementing a regular periodic exchange of short status messages between communicating devices (sometimes called "heartbeat messages"). Failure to receive heartbeat messages from a remote device may indicate a communication problem. Although QoS technologies can enforce communication timing and delivery expectations, they cannot prevent network failures. Because of this, QoS technologies cannot be used as a replacement or a substitute for a plan to respond to communication failures.

In addition to being able to function during a communication failure, a DT should be able to continue safe operation during a cyberattack. Some cyberattacks may not aim to disrupt QoS, but may instead aim to corrupt data or issue harmful commands [37,59,63,92,115]. This means the DT must prioritize data security and must be capable of detecting malicious, falsified, or incorrect data. Some of the major areas of data security include:

- Confidentiality - preventing disclosure of data to unintended recipients
• Integrity - ensuring data has not been modified since it was transmitted or stored

• Authentication - validating the origin, author, or creator of data

Confidentiality is perhaps the most well known area of data security. Confidentiality includes encrypting data before transmitting or storing it. This prevents data from being disclosed to anyone who cannot decrypt it. Integrity is needed to ensure that data has not been modified, whether intentionally or accidentally. This prevents bad data from being used, e.g. corrupted sensor measurements. Authentication can be used to verify the source of data or the identity of a remote actor. This can ensure that only data from the intended source is acted upon, e.g. commands from an authorized operator and not from a malicious user.

Secure hash algorithm (SHA) is a method for generating a cryptographic hash value of some data, with SHA-3 being the most recently approved version [116]. Recipients of data can compare a received hash value against their own calculation of the hash value, and a mismatch between the hash values will indicate that the received data has been modified after the original hash value calculation. The hash value can also be digitally signed using the private key of the originator [117], and the recipient of a signed hash value can use the originator’s public key to ensure the hash value itself has not been altered.

Security certificates such as X.509 [118] can be used verify the identity of a remote actor. A certificate can be signed by other certificates, and a chain of trust can be established through signed certificates. If a trusted certificate has signed another certificate, it is assumed that the signed certificate may also be trusted. Certificate authorities (CA) act as trusted third-parties to sign certificates. A CA may publish a list of revoked/untrustable certificates, known as a certificate revocation list (CRL). An organization or business may maintain their own CA to sign certificates used by their devices or operators.

Security of devices can be improved with cryptographic hardware known as a trusted platform module (TPM) [91, 119–122]. Amongst other things, the TPM will store encryption keys in hardware such that private keys cannot be read out from the TPM. This helps
prevents security keys from being discovered by unauthorized actors. The tradeoff for this approach is that the TPM may become a bottleneck as the cryptographic operations must be performed on the TPM. Use of TPMs can be used to establish trust when communicating with a remote device [119,120], which can address both physical security and data security concerns for cyber-physical systems [91,122,123].

The physics-based model of the virtual twin can be used to detect corrupt or falsified measurements [31,59,64,124]. This idea is also mentioned in [125], but for CPS in general and not specific to DTs. By comparing the received measurements to the virtual twin model, corrupt or falsified data can be detected. This adds a layer of protection as corrupt or falsified data can be detected even if it has bypassed data security measures.

3.2 Real-time requirements

A DT will ultimately be deployed to a live, real-world environment. In order to operate in the real world, a DT must be capable of functioning in real-time. This involves not only the real-time exchange of data between the physical twin and virtual twin, but also the real-time response required from the DT processing system or subsystems. This real-time requirement is a defining DT requirement [16,41,58–60,65,70,80–82,126,127].

The communication networks transferring DT data should be enabled with QoS technologies to help the connected DTs meet their real-time requirements. The processing system or subsystems used to execute the virtual twin and intelligent agent functions may benefit from accelerator hardware such as an FPGA. The real-time requirement must also be taken into account when developing test procedures for a DT. Specifically, a method to test the DT performance in real-time is needed [73,81,126,128].
3.3 Enabling technologies

Tools must be available to support the development of DTs. The physical twin, intelligent agent, and communications layers already have supporting tools from their respective fields. A survey of DT enabling technologies is available in [100]. Physical twin development is similar to existing CHIL development processes. AI/ML tools are readily available for agent development, several of which are open-source. Communication standards and architectures are well established for the communications layer, several of which are also open-source. The concept of a virtual twin is still under development, and likewise, the tools necessary for virtual twin development are not immediately clear.

3.3.1 Physical twin

Real-time simulators and controller hardware-in-the-loop (CHIL) methodologies have proven to be useful technologies when testing controllers which will be deployed in microgrids [129,130]. The ability to test a device in real-time with a realistic level of detail can provide confidence in a device before moving it to later development stages (see Section 4.1).

When developing a DT, a real-time simulator serves as a stand-in of the physical twin. This allows the DT to be developed and tested when an actual physical twin may not be available or when it may be too costly or dangerous to use the actual physical twin. For example, a prototype simulation of a microgrid may be created, and development of the DTs which will be interfacing with it may begin before construction of the microgrid itself. Early incorporation of DTs may provide an advantage to the overall system design process [81].

Real-time simulators may also facilitate reinforcement learning. Reinforcement learning is a trial-and-error process where an intelligent agent learns decide on actions by interacting with its environment and receiving scores for its decisions. For systems where
incorrect actions could disruptive or destructive, use of a real-time simulator will allow a reinforcement learning agent to make mistakes without causing any harm.

Real-time simulators will continue to be an integral part of DT development, much like CHIL methodologies have become commonplace for power electronics development. Their ability to operate in real-time is useful for both testing, data generation, and DT intelligent agent training. Several commercial real-time simulators are available, such as OPAL-RT and Typhoon HIL, making real-time CHIL development and testing an accessible technology.

### 3.3.2 Intelligent agent

AI/ML has also become a highly accessible technology. Tools such as TensorFlow, PyTorch, and MATLAB Deep Learning Toolbox are available to simplify the process of creating and training neural networks. Standards such as the Open Neural Network Exchange (ONNX), Neural Network Exchange Format (NNEF), and OpenVINO have been created to facilitate the storage, exchange, and adoption of neural networks.

The response time of an ML model can be improved by using hardware accelerators. Xilinx Vitis AI and Intel FPGA AI Suite can take a model and generate FPGA acceleration hardware specific to the model. Coral AI provides the Coral Edge TPU (tensor processing unit), which can accelerate TensorFlow models for embedded devices. If the processing system can support it, a graphics processing unit (GPU) may be an option for improved speeds. As the use of DTs expands, these types of hardware accelerators will support their intelligent behavior.

### 3.3.3 Communications

Communication standards and architectures exist to support the level of network QoS required for DTs. IEEE Std 1588-2019 [131] is the standard for precision time protocol
(PTP). PTP is used to synchronize clocks in a network with sub-microsecond accuracy, allowing measurements from throughout the network to be highly time-synchronized. TSN can be used to meet timing requirements in networks with time-critical nature, and technologies such as software defined networking (SDN) can aid efficient and flexible network operation [132,133].

Edge computing [12,19,101] aims to distribute the data and computing capabilities used by a system of devices. In particular, it pushes storage and computing toward the devices at "the edge" of a network. Distributed computing increases the reliability of the system, and network latency can be improved by using more localized communication. Edge AI [134,135] is an adoption of the edge computing paradigm specifically focused on edge computing for AI/ML purposes.

Cloud computing [12,100,101] is another technology which can support distributed, autonomous systems. Cloud-based service providers such as Microsoft Azure and Amazon Web Services (AWS) are publicly available, and organizations may set up their own private cloud computing services. OpenStack is an open-source project for creating and managing cloud computing servers. Cloud-based service providers may include services to supported edge AI, such as Azure Stack Edge and AWS SageMaker Edge.

OPC UA is a service-oriented architecture (SOA) well suited for DT communication [100]. OPC UA was developed by the OPC Foundation and is an open-source, cross-platform, certifiable IEC standard (IEC 62541). OPC UA prioritizes security and contains a flexible information model. The OPC UA information model can allow a DT to describe its capabilities and services to other devices. OPC UA also includes specifications for discovery services, which are services designed to help devices find and connect to other devices. Cloud-based service providers such as Azure and AWS may include support for interfacing with OPC UA devices.

Potential uses of OPC UA for EPS such as smart grids have already been considered [43,136–140]. Mappings between IEC 61850 and OPC UA have been proposed in [43], and
a companion specification for IEC 61850 interoperability is available through the OPC Foundation. Similarly, mappings between CIM and OPC UA have been proposed [141].

OPC UA uses a security up-front approach, making it a suitable option for secure communication in an EPS or smart grid [137,139,142]. An OPC UA extension using TPMs for more secure authentication (see Section 3.1.4) is proposed in [120]. A framework for analyzing the exchange of messages, including OPC UA messages, is proposed in [92] where sequences of messages are analyzed to determine which sequences may put the system in a dangerous state.

The self-describing address space of the OPC UA information model facilitates interoperability [105, 137, 138]. In addition to mapping OPC UA to other EPS communication standards, several works have demonstrated the ability to map the OPC UA information model to ontologies [102,143,144]. Methods for performing automated reasoning on the OPA UA information model are proposed in [102, 145], and [105] uses OPC UA to interface intelligent agents in a multi-agent system (MAS).

### 3.3.4 Virtual twin

Several methods for creating virtual twins have been used. Sections 2.3 and 3.1.2 provide examples, and other virtual twin designs can be found in DT related literature. The tools to help create virtual twins vary, and the appropriate tools are likely to depend on the type of virtual twin model and the system modelling approach being used.

Development of a DT and its virtual twin should ideally take place early in the system design stages [58,69,81]. The tools and methodologies typically used during these design stages, especially those focused on development of physical twins, may be applicable to the development of a virtual twin. The process of creating virtual twins may also be automated, for example as part of a more comprehensive virtual prototyping process (VPP) [146–148]. As the adoption of DTs grows, the tools and methodologies used to facilitate development of virtual twins should continue to be investigated.
Chapter 4

Digital Twin Intelligent Agent
Development Process
4.1 Controller hardware-in-the-loop

Technically readiness levels (TRL) [149] can be used to de-risk a development process. This is especially valuable when dealing with a high financial cost or a system or device whose operation may have an impact on safety. To test devices being deployed in an EPS, CHIL and power hardware-in-the-loop (PHIL) methodologies can assist the advancement through TRL levels 4, 5, and 6 [129]. These steps will require real-time validation [81, 126, 128] and hence require a real-time simulator. Table 4.1 lists technical readiness levels as applied to power electronics. Fig. 4.1 shows how a real-time simulator could interface with a controller through analog-to-digital converters (ADC), digital-to-analog converters (DAC), and digital I/O.

Table 4.1: Technical readiness levels

<table>
<thead>
<tr>
<th>TRL</th>
<th>Description</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>System proven through successful operations</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>System completed and qualified through testing and demonstration activities</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>Demonstration of system prototype in operational environment</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Demonstration of system/subsystem model or prototype in relevant environment</td>
<td>CHIL / PHIL</td>
</tr>
<tr>
<td>5</td>
<td>Component or breadboard validation in relevant environment</td>
<td>CHIL / PHIL</td>
</tr>
<tr>
<td>4</td>
<td>Component or breadboard validation in laboratory environment</td>
<td>CHIL</td>
</tr>
<tr>
<td>3</td>
<td>Analytical/experimental critical function and/or characteristic proof of concept</td>
<td>Offline</td>
</tr>
<tr>
<td>2</td>
<td>Technology concepts and/or applications formulated</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>Basic principles observed and reported</td>
<td>-</td>
</tr>
</tbody>
</table>

4.2 Intelligent agent development process

The DT intelligent agent development process is an iterative process similar to the CHIL development process. The agent development process, shown in Fig. 4.3, has seven significant steps. The first step is to create a simulation which will generate the training data. The second step is to define the inputs/outputs of the DT agent ML model. The third step is to run the simulation and record the training data. The fourth step is to develop the ML model to be used in the DT agent. The fifth step is to train and validate the ML model.
The sixth step is to deploy the DT agent to the controller or processor. The last step before moving to the next stage of technical readiness is to perform real-time CHIL tests on the DT.

An existing microgrid protection design process has been developed and is shown in Fig. 4.2. The aim of the DT agent development process, shown in Fig. 4.3, is to align with this process. This will allow DTs to be developed synergistically with microgrid protection systems.

### 4.2.1 Create simulation

The simulation created in step 1 should be as physically accurate as possible. This may include creating the simulation based on a model-based systems engineering (MBSE) approach [69, 81, 150]. The more accurate the simulation model is, the more closely the
simulated data will represent real-world data. Having data highly representative of a DT’s specific installation will allow it to perform more effectively [99].

The simulation model must be capable of running in real-time on a real-time simulator with a reasonably small time step (the smaller, the better). This create consistency through the development process and allows the DT to be tested against the same scenarios from the simulation.

4.2.2 Define ML inputs/outputs

The ML model inputs and outputs are defined in step 2. The inputs must be values which can be sampled, read, or otherwise acquired from the physical twin in a real-world deployment [62]. The simulation must have signals capable of being recorded to capture the data for the ML model. In a deployment, the input values will most likely be read from ADCs connected to the physical twin sensors while the outputs may be commands for a controller, digital outputs, or signals driven by DACs.

4.2.3 Record simulation data

Step 3 of the agent development process is to run the simulation to generate training data. Every run of the simulation should be slightly different to capture the variety of effects which could occur. For example, data for an electrical fault should be generated at several points along a 60 Hz cycle. This approach is basically a parameter sweep, a technique often used in similar simulation methodologies.

4.2.4 Develop ML model

Development of the ML model takes place in step 4. Development of the ML model involves selecting which model will be used along with the parameters of the model. Development of the model also involves specifying the training loss function or functions.
Details of the model development are dependent on the problem itself, and not all ML models may be suitable for a given problem. If changes to the model inputs or outputs are required, the development process can be repeated from step 1.

4.2.5 Train and validate ML

In step 5, the ML model is trained and validated. The saved simulation data (from step 3) and any other relevant data is used to train the model. The exact approach to training depends on the model, but nearly all approaches will involve trying to minimize a loss function or maximize a reward function. Appropriate training approaches must be followed, e.g. a 70/30 split in the training data where only 70 percent of the data is used to train the model while the remaining 30 percent is used to validate it.

ML training is a time-consuming task and may require several hours, days, or even weeks to complete. If a training process is too slow, it is recommended that the training process first be appropriately timed and profiled. This will provide feedback on what the most time-consuming parts of the process are and what parts could possibly be sped up. The training process of some ML models, e.g. deep learning neural networks, can see a remarkable speedup when a GPU is available to assist with the training. GPUs can take advantage of parallelism in the training calculations, such as large matrix operations.

It is also recommended that information from the training process be saved for documentation purposes. This could include recording loss functions or validation scores over time, periodically saving checkpoints information, and keeping a console output log (or similar) for troubleshooting. Checkpoint information may include the state of the partially trained model, the current training epoch/iteration, and the state of the training data for the epoch/iteration. In the event that the training process fails due to an error or system outage, the checkpoint information can be used to resume the training process from a partially complete state rather than restarting the entire training process.

Because training is a computationally demanding process, storing the training data
and performing the training on the DT itself may not be an effective approach. Instead, the training data and training process may be delegated to another system, possibly an off-site system [12,100,101].

4.2.6 Deploy to controller

The DT agent is deployed to the controller in step 6. The details of this step are also highly problem dependent. However, the DT agent will require processing resources, and the agent cannot be deployed to a controller which does not have these resources or cannot allocate them for the agent.

If the trained agent model cannot be deployed to implementation constraints, the process can be repeated from step 4 to design a more computationally efficient or resource-friendly model. Implementation constraints could include memory usage of the ML model, response time of the ML model when executed on the controller, or FPGA timing constraints for an ML accelerator.

4.2.7 Perform CHIL tests

Step 7 is the last step in the development process. CHIL tests are performed on the controller, including the DT, to validate its real-time performance. The simulation developed in step 1 may be used for the CHIL testing process. This will ensure the DT is tested with the same high level of detail and the same system-wide effects that were captured in the training data.

These tests should test both the performance of the controller functions and the performance of the DT (including the agent). If the CHIL tests are not acceptable, the process can be repeated from step 4 to develop a better performing DT agent. Once the CHIL tests are acceptable, the controller and its DT can move to the next stages of development.
4.3 Summary

Real-time testing is a critical part of DT agent development, and the existing CHIL process can be adopted for testing DT agents. Storage space for the training data is required, along with a system or systems to run the simulations and perform the agent training process. The storage and training systems may be off-site systems and may include accelerator hardware such as GPUs. An example of DT training data generation, storage, and use is shown in Fig. 4.4.

Data measured by a DT may be included in the training data, allowing the DT to improve its operation based on its own experience and observations. The more real-world data that is gathered, the more realistic the training data will be. This will allow the DT to be periodically retrained on the new data with the goal of improving its performance specific to its installation.

In addition to adaptability, the iterative nature of the process can alleviate the burden of updating a DT if the system design changes. For example, if the agent needs to be re-trained following a change to the system design, the process can be resumed from the appropriate step rather than restarting the entire DT development process.

The process can also be repeated as the DT advances through the TRL levels. Later TRL levels will require the DT to be deployed to an operational environment, which will provide new data for the agent to be trained on. If the results of the testing appropriate for a TRL suggest the DT needs to be improved, changes to the DT can begin at the appropriate step of the process while incorporating new information or data, such as data collected from the operational environment.
Figure 4.2: Microgrid protection design process
Figure 4.3: Development process for DT intelligent agent
Figure 4.4: DT training data storage and use cases
Chapter 5

Digital Twin for a Battery Energy Storage System
5.1 BESS DT Development

Fig. 5.1 is a diagram of the layers of the BESS DT. The physical twin is the BESS converter itself. The virtual twin is derived from the converter state-space equations. The intelligent agent contains an action recommendation system to provide recommended actions to the BESS controller based on local measurements. The data communication network connects the BESS to other devices in the nanogrid such as the nanogrid controller or the generator.

During the development of the BESS DT, the first five steps of the DT agent development process (Fig. 4.3) were followed. These steps involved creating the nanogrid simulation, defining the signals to be used by the DT agent, running simulations and saving the data, developing the ML model for the DT agent, and training the ML model using the saved simulation data.

![BESS DT diagram](image)

**Figure 5.1: BESS DT diagram**

5.2 Nanogrid simulation

To generate the data needed for training the BESS DT intelligent agent, an existing simulation of a nanogrid was used and modified as needed. Simulations were run to generate data during various nanogrid configurations and scenarios. The simulation data was
recorded and later used during the training process of the DT agent.

5.2.1 Nanogrid description

The BESS DT is connected to a three-phase AC industrial nanogrid. The nanogrid consists of a diesel generator, a solar PV farm, the battery energy storage system, and three loads. For the case study, it is assumed that the nanogrid is operating in islanded mode. A diagram of the nanogrid is provided in Fig. 5.2.

The diesel generator is rated for 400 KW output at 480 V. The PV farm is rated for 40 KW output at 480 V. The BESS is rated for 200 KW output at 480 V. Two of the loads (Load 1 and Load 3) are low-voltage AC (LVAC) loads. Both LVAC load are 25 KW, one with a power factor (PF) of 0.90 and the other with a PF of 0.95. The other load (Load 2) is a 100 KW line-started induction machine (IM) with 0.80 PF. A list of the ratings is provided in Table 5.1.

Table 5.1: Nanogrid component ratings

<table>
<thead>
<tr>
<th>Component</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generator</td>
<td>500 KVA / 400 KW / 0.8 PF</td>
</tr>
<tr>
<td>PV</td>
<td>45 KVA / 40 KW / 0.9 PF</td>
</tr>
<tr>
<td>BESS</td>
<td>222 KVA / 200 KW / 0.9 PF</td>
</tr>
<tr>
<td>Load 1 (LVAC)</td>
<td>25 KW / 0.90 PF</td>
</tr>
<tr>
<td>Load 2 (IM)</td>
<td>100 KW / 0.8 PF</td>
</tr>
<tr>
<td>Load 3 (LVAC)</td>
<td>25 KW / 0.95 PF</td>
</tr>
</tbody>
</table>

Five fault locations were used (F2 - F6 in Fig. 5.2). Eleven fault scenarios were simulated at each of these locations. A list of the fault types is provided in Table 5.2, and a description of the fault locations is provided in Table 5.3.

5.2.2 Data generation

Step 1 of the agent development process (Section 4.2.1) is to create a physically accurate simulation. The nanogrid simulation used for the case study takes a MBSE approach
This allows devices to be tested more thoroughly by simulating their interaction with the entire system rather than simply with a testbed. A high level of simulation detail along with a low simulation timestep creates a more physically realistic simulation. This level of detail creates more realistic data for both test purposes and for training the DT agent.

The nanogrid simulation was created in Simulink and is run with a timestep of 1 microsecond. The simulation has the possibility of being run on a real-time simulator such as OPAL-RT. The ability to run the simulation on a real-time simulator creates the opportunity to perform real-time testing on the DT later in its development process (specifically in step 7, Section 4.2.7).
Table 5.2: Simulation fault types

<table>
<thead>
<tr>
<th>Fault type</th>
<th>Phases involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line to line</td>
<td>A-B, B-C, or A-C</td>
</tr>
<tr>
<td>Line to ground</td>
<td>A-Gnd, B-Gnd, or C-Gnd</td>
</tr>
<tr>
<td>Line to line to ground</td>
<td>A-B-Gnd, B-C-Gnd, or A-C-Gnd</td>
</tr>
<tr>
<td>Three line</td>
<td>A-B-C</td>
</tr>
<tr>
<td>Three line to ground</td>
<td>A-B-C-Gnd</td>
</tr>
</tbody>
</table>

Table 5.3: Simulation fault locations

<table>
<thead>
<tr>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>Fault by generator</td>
</tr>
<tr>
<td>F3</td>
<td>Fault on bus</td>
</tr>
<tr>
<td>F4</td>
<td>Fault by load 1 (LVAC)</td>
</tr>
<tr>
<td>F5</td>
<td>Fault by load 2 (induction machine)</td>
</tr>
<tr>
<td>F6</td>
<td>Fault by load 3 (LVAC)</td>
</tr>
</tbody>
</table>

Step 2 of the agent development process (Section 4.2.2) is to define the ML inputs and outputs. The simulation was modified from its original state to accommodate this step of the ML development process. The virtual twin (discussed in Section 5.4) was implemented in the simulation. Signals were added to log the necessary data for the intelligent agent (discussed in Section 5.5).

Step 3 of the agent development process (Section 4.2.3) is to run the simulation and record data. To generate data for training the BESS DT agent, the simulation was divided into two phases. The first phase ran the simulation from a black start into a configuration. The simulation state was saved after the configuration had reached steady state. The second phase simulated scenarios, which were events occurring in the nanogrid such as faults or load steps. A diagram of the simulation setup and the data generation process is provided in Fig. 5.3.

A configuration is the state of the nanogrid, and different configurations have the loads and generators connected or disconnected in different fashions. Each scenario resumed the simulation from a saved configuration state. This approach generated data for
the scenario events under a variety of load and generation configurations.

Data was generated using eight configurations, where each configuration was followed by approximately 30 - 60 scenarios (depending on the relevance of the scenario to the configuration). The scenarios simulated events with starting times spaced uniformly across 200 points of the first scenario 60 Hz cycle (83.33 microseconds apart).

These simulations were carried out on Mortimer, a high performance computing (HPC) cluster available to researchers at the University of Wisconsin - Milwaukee. Each simulation instance can be run independently, which allows the data generation process to take advantage of process level parallelism. Specifically, a large number of simulations (e.g. possibly up to 200 on Mortimer) were run simultaneously. Simulation data for each configuration and scenario was saved in MATLAB MAT files. Approximately 55,200 MAT files were created with about 250 GB of raw simulation data. The generated simulation data was saved on a storage node of Mortimer and was copied to an off-site solid-state hard drive.

Step 4 of the agent process (Section 4.2.4) is to develop the ML model(s). After capturing some trial data, it was determined that the original simulation needed to capture more data for training the DT agent. This required a change to the simulation model, which was in iteration back to step 2 of the process. Trial data for the revised simulation was deemed sufficient, and the full set of simulations was run (step 3). Future modifica-
tions to the BESS DT could repeat these steps to generate data for a new ML model.

Once the nanogrid simulations had completed, the MATLAB Deep Learning Toolbox was used to develop the ML models to be used by the BESS DT agent. The raw simulation data was first pre-processed to contain only data relevant to the DT, then was used by the ML training process to train the BESS DT agent. Details of the agent are provided in Section 5.5 and Appendix B.

5.3 Physical twin

The BESS DT physical twin is the BESS converter itself. The BESS converter topology is a 3-phase, 2-level voltage source converter (VSC). The VSC has an inductor-capacitor-inductor (LCL) filter with passive resistive damping in series with the capacitors. The VSC is connected to a wye-delta transformer. The wye side of the transformer is connected to the nanogrid bus through a transformer. The delta side of the transformer connects to the converter. The wye side is grounded for electrical safety and ground fault protection. A schematic of the BESS converter can be found in Fig. A.1.

The BESS simulation is a full switching model of the converter. This level of detail helps capture effects which will occur within a single switching cycle and creates a highly accurate physical simulation (step 1 of the agent development process). This also requires that the simulation timestep or the physical twin sampling period be small enough to capture these effects.

Several considerations went into the choices of physical twin measurements. The chosen measurements are similar to what typical control systems would use and to what many of the works in Sections 2.2 and 2.3 used. The following measurements are taken from the BESS:

- Filter inductor currents
- Filter capacitor voltages
• Output inductor currents
• Grid voltages
• Switch states

Currents and voltages are measured from their corresponding components for each of the three phases. Grid voltages are measured for all three phases, and each of the six switch states are recorded, giving a total of 18 physical twin measurement channels. These measurements are sampled at a rate of 100 KHz. The converter switching frequency is 10 KHz. A sampling rate of 100 KHz is high enough to capture effects within the 10 KHz switching frequency.

5.4 Virtual twin

The approach to the BESS DT virtual twin was heavily inspired by [64] and [65]. The BESS virtual twin model is derived from the state-space equations for the BESS converter. These equations can be arranged in the standard \( \dot{x} = Ax + Bu \) format of a state-space model. This model will scale with \( O(N^2) \) complexity given \( N \) physical twin measurement channels (an approach to reducing complexity is provided in [65]).

The BESS simulation was created in the CSEES lab, so all its details (component values, ratings, etc.) were available. This allowed the virtual twin model to be created in a white-box fashion. The \( A \) and \( B \) matrices are known and both the state vector \( x \) and the input vector \( u \) can be measured at each timestep. An example plot of output inductor current measurements and predictions is shown in Fig. 5.4.

This model meets the requirements for a virtual twin:

• It is based on the differential equations governing the physics of electric circuits, hence it is physics-based
The calculations of the model are matrix operations, making it computationally efficient and real-time capable.

By combining the state-space model with the forward Euler method, the model can predict the BESS converter state at the next timestep.

The error vector $\epsilon_t$ for timestep $t$ is calculated as the difference between the predicted state value $\hat{x}_t$ and the actual (measured) state value $x_t$. Since the state consists of 9 measurements (filter inductor currents, filter capacitor voltages, and output inductor currents), the error vector contains 9 components, one for each of the state measurements. Fig. 5.5 shows a view of the virtual twin measurements and predictions for the phase A output inductor current, zoomed in on a very small timescale. The mismatch between the predicted values (dots) and measured values (the line) is visible in the figure, and this mismatch would create the error vectors.

Fig. 5.6 illustrates some virtual twin measurements and their error vectors. Fig. 5.6a shows filter capacitor voltage measurements and Fig. 5.6b shows output inductor cur-
rent measurements. The $L^2$ norms of the error vector components for the filter capacitor voltages and the output inductor currents are shown in Figs. 5.6c and 5.6d, respectively. More examples of measurements and error vectors can be found in Appendix A.

Both [64] and [65] use the error vector to detect and identify faults. Hard thresholds are set on each component of the error vector in [64] while a probabilistic approach to the setting threshold level is used in [65]. Faults are identified in [64, 65] by using an inner product to compare recent error vectors to a predefined library of example fault error vectors.

The error vectors for the BESS DT are used by the intelligent agent. This presents the possibility of detecting and identifying faults without the use of thresholds on the error vector. This also allows different ML models to be created to achieve different goals (e.g. internal monitoring) based on the same input.

The virtual twin model is implemented in the simulation. Data for the virtual twin is sampled at 100 KHz (a timestep of 10 microseconds). For deployment, this virtual twin model could be implemented on an FPGA alongside the controller for the BESS converter.
Details of the virtual twin model can be found in Appendix A.

### 5.5 Intelligent agent

The goal for the DT intelligent agent is to respond to events in the nanogrid by recommending actions to the BESS controller. The sensors for the agent are the physical twin sensors (described in Section 5.3). The actuator for the agent is the recommended action, which will cause the BESS to change its operation. The "decide" part of the sense-decide-act loop involves using the physical twin measurements and the error vector (from the virtual twin) as input to the agent ML model model to get a recommended action.

The actions the agent can recommend are: ride through, disconnect, and switch to grid forming. The agent can also recommend that no action be taken if no action is needed. A list of the events and their actions is provided in Table 5.4. The BESS should take no action
if there is a load step or if no events are occurring. If there is a fault at one of the loads, the BESS should ride through the fault. If there is a fault on the bus, the BESS should disconnect. If there is a fault at the generator or the generator loses its prime mover, the BESS should switch to grid forming. These actions were prescribed by an expert based on their knowledge of the nanogrid and its operational goals (including resilience).

Table 5.4: BESS DT agent actions

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load step</td>
<td>No action</td>
</tr>
<tr>
<td>Fault at loads (F4, F5, F6)</td>
<td>Ride through</td>
</tr>
<tr>
<td>Fault on bus (F3)</td>
<td>Disconnect</td>
</tr>
<tr>
<td>Fault by generator (F2)</td>
<td>Switch to grid forming</td>
</tr>
<tr>
<td>Loss of generator prime mover</td>
<td>Switch to grid forming</td>
</tr>
</tbody>
</table>

A noteworthy difference between the BESS DT and the DTs described in [64, 65] is the type of faults and events being detected/identified. The BESS DT agent aims to detect external events (events occurring in the nanogrid) while [64, 65] focus on internal events such component degradation. However, the use of this BESS DT agent does not preclude the possibility of another coexisting DT agent whose purpose is to monitor the BESS itself in a manner similar to [64, 65].

Step 4 of the agent development process (Section 4.2.4) is to develop the ML model. Development of the BESS DT agent model consisted of creating the neural networks used in the model. The agent model consists of several deep learning neural networks, and the output of one of these networks is a code representing the recommended action.

The action to be taken in response to an event is based on the event itself. This means the agent must be capable of detecting and identifying events in the nanogrid. For example, the agent must be able to distinguish between a fault at the generator (F2) and a fault on the bus (F3) to recommend the correct action (switch to grid forming or disconnect, respectively). In a general sense, this makes the action recommendation system a classifier, i.e. it classifies the measurements and error vectors to a specific action.
The inputs for the agent model were defined in step 2 of the agent development process (Section 4.2.2). Specifically, they were chosen to be the error vector and all physical twin measurements except the switch states. This creates a total of 21 inputs to the agent model on each timestep.

Use of current and voltage measurements as inputs to the agent model is similar to many of the works in Sections 2.2 and 2.3. These are physical measurements from the real world in which the DT is operating, so they are a natural choice for input to the agent model.

The agent model consists of three deep neural networks (DNN). Two of the network are an encoder and a decoder, which together form an autoencoder. The last network is the action recommender. The model uses time series sequences of input vectors containing the physical twin measurements and error vector components. The model operates a minimum sequence length of one 60 Hz cycle worth of data. This sequence can be a moving window, meaning that the model can operate on a stream of incoming data and can recommend actions multiple times within a cycle (i.e. not simply once per cycle).

The action recommendation network is directly responsible for recommending the actions. The input to the action recommendation network is taken from the latent space of the autoencoder, i.e. the bottleneck characteristic to autoencoder topologies. The main intent for using the autoencoder is to provide dimensionality reduction for the action recommended. Another popular use of autoencoders is anomaly detection [151]. Although the autoencoder in the BESS DT is only used for dimensionality reduction, it could also be adopted for anomaly detection purposes (see Section 7.2).

The latent space of the autoencoder was selected to be 6 dimensions. The encoder maps the 21 inputs to this 6-dimensional space while the decoder aims to output a 9 component reconstruction the input error vector components. This means the autoencoder must try to embed 9 dimensions of information into a 6-dimensional space. To achieve the best reconstruction, the autoencoder must learn which information is most relevant.
to the reconstruction and discard information which has little to no impact on the recon-
construction. The points in the 6-dimensional latent space are used as features of the
physical twin measurements, reducing the information from 21 dimensions of raw data
to 6 dimensions of feature data.

The autoencoder is a variational autoencoder [152]. In a variational autoencoder, the
encoder outputs parameters for a distribution on the latent space. The distribution and
parameters are used to sample a point in the latent space, and this point is used by the
decoder to reconstruct the input. Often, a Gaussian distribution is used and every dimen-
sion is assumed to be independent (no covariance between dimensions). This means the
encoder outputs two N-dimensional vectors, one to be used as the mean parameter and
the other to be used as the diagonal of the covariance matrix parameter.

Sampling from the latent space distribution can facilitate a Monte Carlo approach to
training where the decoder can attempt to reconstruct the same input using several dif-
ferent latent space samples. This approach pushes the decoder to learn from regions of
the latent space (i.e. points sampled from the distribution) rather than from exact, spe-
cific points. During deployment, the model can simply use the mean parameter from
the encoder as the latent space point. The mean is the maximum likelihood estimator
(MLE), making the mean the point most likely to be sampled from the distribution and
thus bypassing the need for sampling during deployment.

This sampling approach is also used to train the action recommender. The action
recommender performs a 16x downsampling on its input sequence (a sequence of points
from the latent space). Similar to having the decoder reconstruct the same input across
several sampled points, the action recommender is trained to recommend the same input
across several sampled points. The intent of this is similar to its intent for training the
decoder, i.e. it aims to have the recommender learn from regions of the latent space rather
than from specific points.

Many of the layers in the networks are convolutional layers. Convolutional layers
were used by the networks in [50, 51] for fault detection and identification. These layers are based on the convolutional operation and can adapt their weights to learn different convolutional filters. Layers in the network also include "skip connections", a technique used in residual neural networks (ResNet) [153].

As noted in [54, 62], the agent model must capture temporal dependencies in the input data. The convolutional layers in the agent operate along the time dimension of the input data. This approach causes the convolutional filters to span several time points, thus capturing temporal relationships in the filter output.

The action recommendation network also includes a LSTM layer. LSTM and similar RNN layers are used in [48, 49, 51, 52]. LSTM layers can capture long-term dependencies in input sequences (including temporal dependencies in the case of the BESS DT). Many of the notable effects in the error vectors are transient, short-lived effects which are most prominent during event inception. Since the action recommender needs to recommend actions during the event and not simply at its start, the LSTM layer was added to retain the effect of previous inputs and to provide more accurate recommendations even after the error vector effects have subsided.

Step 5 of the agent development process (Section 4.2.5) is training the agent and validating its performance. The BESS DT agent was trained on the simulation data (described in Section 5.2.2). Measurements and error vectors from each scenario were provided to the model along with the action to be recommended for the scenario.

The model was trained using three terms in its loss function. The first term was cross-entropy loss, a popular loss function used by classifiers. The cross-entropy loss quantifies the loss of the action recommender outputs. The second term loss term is the decoder reconstruction loss. The mathematical model of the reconstruction problem typically uses a Gaussian form, and Gaussian log-likelihood is used simplify the loss calculation. Minimizing the Gaussian log-likelihood loss reduces to the same calculation as mean squared error (MSE), hence MSE is used as the reconstruction loss. Finally, a Kullback–Leibler (KL)
divergence term (sometimes known as "relative entropy") is used to regularize the distributions on the latent space [152]. Specifically, the KL divergence between the distribution defined by the encoder outputs and a standard Gaussian distribution is used as the KL divergence loss. An approach to balancing the KL loss function with other loss functions is provided in [154]. A similar approach to balancing the KL divergence term was taken while training the BESS DT model.

Once the agent was trained, its performance was evaluated using a confusion matrix. A confusion matrix quantifies the performance of a classifier by comparing the classes output by the classifier with the correct classes. For the BESS DT, the confusion matrix compares the actions recommended by the BESS DT to the actions which should have been taken for the corresponding scenarios. The resulting confusion matrices for the BESS DT are discussed in Section 6.2.1.

MATLAB Deep Learning Toolbox was used to create and train the BESS DT agent neural networks. At a sampling frequency of 100 KHz, the model requires a minimum sequence of 1667 input vectors. Using 16x downsampling in the action recommender means recommendations are available every 16 samples, i.e. roughly every 1.04 milliseconds or at a rate of 960 Hz. Details of the model can be found in Appendix B.

5.6 Data communications

The purpose of the BESS DT agent is to operate without active communications, so a communication method was not developed for the BESS DT. It should be noted that this lack of attention to a communication method is simply due to a scope limitation of this work. A fully developed and deployed BESS DT should have a communication method. A discussion of communication for DTs can be found in Section 3.1.4, and recommendations for future work on BESS DT communication can be found in Section 7.2.
Chapter 6

Results
6.1 Training

The BESS DT was trained on the previously generated simulation data (Section 5.2). This process involved evaluating and updating the agent ML model using the simulation data. The model was saved after the training process for evaluation and possible future deployment.

6.1.1 Training process

Training of the BESS DT took place on Mortimer, the same system used to generate the simulation data in Section 5.2.2. Mortimer includes two GPU nodes, both equipped with an NVIDIA A10 (Section 6.1.3). MATLAB scripts were developed to read training data from its storage location on Mortimer, arrange the data for the training process, evaluate and update the DT agent ML model, and save the results.

The training process evaluated the agent model using mini-batches of training data (Section 6.1.2). The loss functions were used to determine the performance of the model and to get the training gradients. Losses were recorded over time to document the training process (Section 6.1.4). Training gradients were calculated from the losses, and adaptive moment estimation (ADAM) [155] was used to update the model.

The training process used a 70/30 split on the simulation data [156]: 70% of the data was used for training while the remaining 30% was used for validation. The training and validation data were shuffled before being split into mini-batches. A training iteration consisted of using a single mini-batch to evaluate and update the model. A training epoch consisted of all iterations (i.e. all mini-batches) required to evaluate and update the model on all the training data. The model was trained for 50 epochs.

Validation occurred periodically throughout each epoch. Validation was performed on the validation data, which was previously separated from the training data. This means the model was evaluated on data that was not present in the training data. The
validation data was passed to the model and the loss scores were used as the validation metric. The model which received the best validation score (i.e. the lowest validation score) was saved. In addition, the model was saved after each epoch and saved periodically as checkpoint data (see Section 4.2.5).

Final evaluation of the model was performed using a confusion matrix (Section 6.2.1). Section 6.3 contains plots and diagrams to illustrate and interpret the BESS DT agent performance. A discussion of the results is in Section 6.4.

### 6.1.2 Mini-batches

The agent model was trained on mini-batches of data [157]. Each mini-batch consisted of 64 training examples, where each example contained 6 cycles of scenario data (see Section 5.2.2) along with the scenario label and the event inception time. The mini-batch size of 64 was chosen empirically based on memory constraints.

### 6.1.3 Use of a GPU

As mentioned in Section 4.2.5, using a GPU can significantly improve training time. Use of a GPU was essential to the training process for the BESS DT. Using the A10 GPU, the training time for 50 epoch was roughly 5 days. Without a GPU, the projected training time for 50 epochs on an Intel i5-1135G7 and 8 GB of RAM was several weeks. MATLAB Deep Learning Toolbox contains features to automatically handle data conversion and data transfer to/from the GPU, which simplified the training process and created portability between a local, non-GPU development system and the GPU nodes on Mortimer.

### 6.1.4 Loss functions

The loss function of the model contains three terms: reconstruction loss, action recommendation loss, and a regularization loss. The reconstruction loss is calculated as the
MSE between the reconstructed components and their corresponding inputs. The action recommendation loss is the cross-entropy loss between the recommended action and the correct action. The regularization loss is the KL divergence between the latent distribution and a standard Gaussian distribution. The regularization term was weighted as a ratio of the best reconstruction and action losses to the reconstruction and action losses for the current mini-batch, similar to the approach in [154].

Validation loss plots for the BESS DT agent training are shown in Figs. 6.1 and 6.2. Fig. 6.1 shows the reconstruction loss and Fig. 6.2 shows the action recommendation loss. These losses are the total loss across the mini-batch (i.e. not normalized for mini-batch size). Both figures show the losses decreasing during training. The reconstruction loss stop decreasing noticeably about half way through the 50 epochs, and while the action recommendation loss stops decreasing significantly, it may have benefited slightly from more training. Fig. 6.3 shows the KL divergence loss. The KL divergence loss is a regularization term [152] intended to keep latent points near the origin rather than scattered throughout the latent space. This makes the KL divergence loss an auxiliary term, and it’s decrease is less important than the decrease of the reconstruction and action recommendation losses.

6.2 Evaluation

After training, the performance of the BESS DT agent was measured and demonstrated. Confusion matrices were used to measure the accuracy of recommended actions, and plots were created to demonstrate the agent providing recommendations in response to nanogrid events.
6.2.1 Confusion matrices

The performance of the agent model was evaluated with confusion matrices (Section 5.5). The confusion matrices were created using post-event action recommendations. Confusion matrices were created from validation data consisting of one, two, and three cycles of post-event data.

Scenario data was loaded and trimmed to the time window appropriate for the corresponding confusion matrix. The evaluation data was passed to the agent model, and each action recommended was compared to the correct action. The confusion matrices were created by tallying the correct and incorrect recommendations and normalizing the counts to percentages across each row.

Some actions may be recommended in response to events with distinctly different electrical characteristics. For example, a switch to grid forming is recommended for both a generator fault and a loss of generator PM. A fault will cause substantial fluctuation on the nanogrid bus while a PM loss will be a subtle and slow-moving change. To assist
the DT agent in these cases, several actions have been split into recommendations along with their reason. The "switch to grid forming" action is split into PM loss events and generator fault events. LVAC load steps and steady-state operation (no events) comprise one type of the "no action" recommendation while an IM load step is the other "no action" recommendation.

Each row of the confusion matrix corresponds to the correct action, and each column corresponds to the recommended action. For a given row, off-diagonal values show how frequently the agent failed to correctly recommend the action corresponding to the row. For a given column, off-diagonal values show how frequently the agent incorrectly recommended the action corresponding to the column. Percentages in the figures are truncated to a single digit, so entries showing "0.0%" had a non-zero count which amounted to a very low percentage. A perfect evaluation would create a confusion matrix with 100% on the diagonal and 0% everywhere else.

Fig. 6.4 shows the confusion matrix for action recommendations during one cycle of
post-event data, Fig. 6.5 shows the confusion matrix for two cycles, and Fig. 6.6 shows the confusion matrix for three cycles. The row-normalized accuracy of correctly recommended actions is summarized in Table 6.1 for one, two, and three-cycles of post-event data.

Table 6.1: Correct recommendations post-event

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Disconnect (bus fault)</th>
<th>Grid forming (PM loss)</th>
<th>Grid forming (generator fault)</th>
<th>No action (no event, LVAC load)</th>
<th>No action (IM load)</th>
<th>Ride thru (load fault)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>95.3%</td>
<td>58.5%</td>
<td>93.6%</td>
<td>66.2%</td>
<td>99.9%</td>
<td>99.6%</td>
</tr>
<tr>
<td>2</td>
<td>90.2%</td>
<td>77.6%</td>
<td>93.2%</td>
<td>71.0%</td>
<td>99.9%</td>
<td>98.9%</td>
</tr>
<tr>
<td>3</td>
<td>95.2%</td>
<td>95.7%</td>
<td>90.0%</td>
<td>85.3%</td>
<td>98.1%</td>
<td>99.6%</td>
</tr>
</tbody>
</table>

Figure 6.3: KL divergence loss
6.3 BESS DT demonstration plots

To demonstrate the actions recommended by the BESS DT, plots were created to show the nanogrid bus voltage alongside the recommendations over time. The correct action is denoted by the orange line while the action recommendations are plotted as blue dots. The event inception time is indicated by a vertical dashed line. Plots of reconstructed error vectors are provided in Appendix B.

Figs. 6.7, 6.8, and 6.9 show recommended actions in response to a line-to-line (phase A-B) fault on the bus (F3), at the generator (F2), and at the induction machine (F5), respectively. Figs. 6.10, 6.11, and 6.12 show recommended actions in response to a three-line-to-ground fault on the bus, at the generator, and at the induction machine, respectively.
Fig. 6.13 shows the recommended actions following a load step at one of the LVAC loads (load 3). Fig. 6.14 shows the recommended actions following a load step at the induction machine (load 2).

Figs. 6.15, 6.16, and 6.17 show action recommendations in response to a generator loss of prime mover. A loss of the prime mover is a slow moving event. Grid voltage and frequency will begin to decrease as the PM transfers the remaining energy from its angular momentum into the EPS. If another energy source (e.g. the BESS) does not assume the responsibility of grid former, the grid will lose stability.

Fig. 6.15 shows the action recommendations for one cycle after a PM loss. Within one cycle, the agent does not detect the PM loss. Fig. 6.16 shows the actions for two cycles after a PM loss. After two cycles, the agent has detected the PM loss and begins...
Figure 6.6: Confusion matrix, three cycles post-event

to recommend the "switch to grid forming" action. Fig. 6.17 shows the actions for nine cycles after a PM loss. After nine cycles, the majority of the actions recommended by the agent are "switch to grid forming". This effect can also be seen in the confusion matrices as the accuracy of correctly responding to the PM loss increases from 58.5% in the one-cycle matrix to 95.7% in the three-cycle matrix.

6.4 Discussion

Based on the confusion matrices and demonstration plots, the performance of the BESS DT agent is satisfactory. The agent can detect events in the nanogrid and recommend actions with a high degree of accuracy, and it can do so using only local information (i.e. 
IEEE Std 1547-2018 specifies "permissive operation" for a 160 millisecond window while the bus voltage falls between 0.7 and 0.5 volts per unit (see the second-to-last row in Fig. 1.2). This 160 millisecond window is equivalent to the time of ten 60 Hz cycles. The bus voltage may drop for several reasons, including a line-start of the IM load or a loss of the generator PM.

The BESS DT agent can detect both an IM load step and a loss of PM within nine cycles or less, and it can do so possibly even before the bus voltage falls below 0.8 volts per unit threshold. This leave the BESS time to take the appropriate action (e.g. switch to grid forming following a PM loss) before the 160 millisecond window expires.

Some events lead to lower performance than others, such as a three-line-to-ground fault on the bus (Fig. 6.10) versus at the generator (Fig. 6.11). Events with less prominent effects can also lead to incorrect recommendations, such as a briefly recommending a switch to grid forming following a LVAC load step (Fig. 6.13) or failing to detect a loss of PM within the first cycle (Fig. 6.15).

From the confusion matrices, it can be seen that the "no action" scenarios (i.e. the "no action" row in the confusion matrices) have the most frequent incorrect recommendations. Bus faults (F3) and generator faults (F2) can also be challenging to distinguish, with the BESS DT occasionally recommending the action for one scenario when the other scenario is actually occurring. This is the first version of the BESS DT agent, and although its performance is satisfactory, its could most likely be improved. Potential improvements to the BESS DT agent can be found in Section 7.2.

The creation of this BESS DT intelligent agent is meant to be a demonstration of where AI/ML fits into the DT architecture and to demonstrate how the BESS DT can be used to take action in response to events in the nanogrid. The physical twin provides measurements to the virtual twin. The virtual twin calculates an error vector using its physics-based model of the measurements from physical twin. The error vector and measure-
ments are used as inputs to the DT intelligent agent, and the agent recommends actions to be taken by the physical twin.

Most importantly, the BESS DT obtained these results without a communication network. This presents proof-of-concept that the DT would be able to recommend actions even while the communication network has failed.

Figure 6.7: Recommendations, line-to-line (A-B) fault on bus
Figure 6.8: Recommendations, line-to-line (A-B) fault at generator

Figure 6.9: Recommendations, line-to-line (A-B) fault at IM
Figure 6.10: Recommendations, three-line-to-ground fault on bus

Figure 6.11: Recommendations, three-line-to-ground fault at generator
Figure 6.12: Recommendations, three-line-to-ground at IM

Figure 6.13: Recommendations, LVAC load step
Figure 6.14: Recommendations, IM load step

Figure 6.15: Recommendations, one cycle after PM loss
Figure 6.16: Recommendations, two cycles after PM loss

Figure 6.17: Recommendations, nine cycles after PM loss
Chapter 7

Future Work
7.1 Implementation

The BESS DT is designed to use a sampling frequency of 100 KHz. This means both the virtual twin and the agent must be capable of operating on a stream of data being input at this rate. Details of the implementation are open-ended, but constraints imposed by the real-time requirements and the sampling frequency must be satisfied.

The ML model used by the BESS DT agent provides action recommendations roughly every 1.04 millisecond (see Section 5.5). To implement this model, the processor executing the ML model needs a response time of less than 1.04 milliseconds. As noted in Section 7.2, a more efficient model may be able meet this timing requirement more easily than the current model and may be able to do so with comparable accuracy.

The real-time requirements of the virtual twin must be taken into account when choosing where or how to execute the virtual twin model. Implementing the model in an FPGA can create deterministic timing but will incur resource demands. A real-time processor may also be acceptable if the processor can meet the timing demands.

The BESS DT agent will be re-trained as it collects data from its nanogrid [99]. Considering the processing and memory requirements generally needed for ML training, the training will likely take place on an off-site system. This system must include storage for the collected nanogrid data and the computational capability to re-train the model in an acceptable amount of time. Requirements for this system should include consideration towards how much storage space will be needed to cover the lifetime of the DT.

MATLAB and Simulink have several development tools supporting the Deep Learning Toolbox. Simulink can implement trained deep learning models in its simulation models. This allows offline experimentation and validation by allowing the ML model to interact with the system simulation. For deployment, MATLAB includes code generation tools which can implement the model in C/C++, HDL, and CUDA. This creates options for deploying the ML model and allows it to be deployed to a processor, an FPGA or system-on-chip, or a CUDA-capable GPU. MATLAB also contains a function to export
deep learning models to TensorFlow. As noted in Section 3.3, support for implementing TensorFlow models is gaining adoption among hardware vendors.

7.2 Improvements and related applications

Currently, the ML model only provides a recommended action for the inputs, and it does so without explicitly classifying which event is occurring. The model could be improved to classify events individually. The ability to identify each event would be useful for post-fault recovery and other coordination tasks. This would also aid technicians who may need to diagnose a problem in the EPS. For reasons such as these, identifying specific events may be useful for the recoverability and repairability stages of the resilience curve (Fig. 1.1).

No substantial experimentation was performed with downsizing the networks in the BESS DT agent model or with using other network topologies. It is likely that the performance of the model could be improved or made more efficient. For example, model performance could be evaluated using varying levels of network depth, e.g. more or fewer repeated residual blocks. Other topologies, such as different arrangements of the convolutional and LSTM layers, could be evaluated as well.

A popular use of autoencoders is anomaly detection. The autoencoder is trained to reconstruct expected, normal data. When presented with anomalous data, the autoencoder will be unable to accurately reconstruct the data. A metric such as MSE can be used to measure the reconstruction error. Normal data will be reconstructed with a low MSE while anomalous data will be reconstructed with a higher MSE.

The use of this approach to anomaly detection may be useful when interfacing the nanogrid with a larger EPS or a mesh of nanogrids. The model could be trained on all the events which would occur in the nanogrid itself. Events occurring outside the nanogrid, e.g. in a neighboring nanogrid, would appear as anomalous data. Identifying
unrecognized data as anomalous could be helpful for recommending actions to connect or disconnect from neighboring nanogrids.

During nominal communication conditions, the communication network can be utilized to coordinate resilient operation [8] or fault recovery [38]. As mentioned earlier, a ML model could provide event identification, and this information could be shared with other controllers in the nanogrid. The usefulness of this DT under more relaxed communication conditions could be studied, e.g. while the communication network is functioning normally.

As mentioned in 5.5, a DT agent could also monitor the BESS for internal faults or degradation. The health of the BESS could be assessed under the same faults and conditions used in this work. This information could be useful for preventative maintenance or for system design, e.g. evaluating a simulated physical twin’s health under off-nominal conditions.

Various methods of creating a virtual twin should be explored. For example, the process of creating the BESS DT virtual twin was a white-box approach. Sufficient data for a white-box approach may not always be available, so gray-box and data driven approaches to creating virtual twins should be developed. The virtual twin may also be created using information from another development process, e.g. a VPP.

Reinforcement learning is an ML approach where an intelligent agent performs an action or series of actions and receives a utility value (a numeric score or a "reward") based on its chosen actions. The goal of the agent is to learn which action(s) will maximize its utility value. The training process is repeated while the agent learns through "trial-and-error" by adapting its choice of actions based on the utility value.

Reinforcement learning will play a central role in the development of DTs. When the DT performs an action, it’s effect will change the data (the physical twin measurements) being observed by the DT. Reinforcement learning works naturally with this closed-loop interaction. Techniques from reinforcement learning such as proximal policy optimiza-
tion (PPO) [158] can be used to learn which actions are "good" actions when the correct actions are not prescribed in advance. "Actions" may be similar to the BESS DT actions described in this work, but actions may also include actively changing controller setpoints to modify controller behavior.

The specific utility values used for reinforcement learning (and other ML techniques) are highly problem dependent. In the context of EPS and resilience, feedback needs to be provided on "how resilient" an action is. Resilience metrics and the ability to assign a resilience score to the state of an EPS or to an action taken by a DT will be highly beneficial. This will allow DTs to be developed which can interact with their environment (including a real-time CHIL environment) to learn which actions will maximize resilience.
Chapter 8

Summary
8.1 DT architecture

A novel DT architecture has been proposed in Chapter 3. The DT architecture consists of four layers: physical twin, virtual twin, intelligent agent, and communications. The physical twin consists of sensors and actuators to interact with a real-world device or system. The virtual twin is a real-time digital model of the physical twin. The intelligent agent is responsible for learning and decision making. Communications is responsible for the secure exchange of data with other devices or human operators.

This DT architecture serves as a bridge between the real world and a digital system. The physical twin interfaces with the physical environment while the virtual twin provides the expected behavior of the physical twin. The agent analyzes data from the virtual twin and recommends actions for intelligent control of the physical twin. The communications layer connects the DT to the rest of the digital system and allows the DT to interact with other devices or human operators.

Since the DT is intended to interact with a real-world environment, it must be capable of operating in real time. This real-time requirement sets constraints on the DT layers, all of which must have the processing capability to respond in an appropriate amount of time.

8.2 DT agent development process

An iterative process for developing a DT intelligent agent has been proposed in Chapter 4. The iterative process allows the agent to be modified, re-trained, and improved during both development and deployment. The process allows the incorporation of new information and data, such as data recorded from the environment in which the DT has been deployed.

A DT will benefit from a development and testing process which can be carried out simultaneously with other system design processes, such as a microgrid protection design
process. The DT’s ability to operate in real time must be tested, and this need is taken into consideration in the DT agent development process. Creation of a real-time capable simulation in the early steps is recommended, allowing the same simulation to be used for the required real-time CHIL testing in the later steps.

8.3 BESS DT

The DT architecture and agent development process were followed to create a DT for a BESS. Details of the BESS DT are provided in Chapter 5 and Appendices A and B. The physical twin is the power converter of the BESS. The virtual twin was created using a state-space model of the BESS converter. The intelligent agent contained several deep learning neural networks to recommend actions based on physical twin measurements and virtual twin error vectors. The communications layer was deliberately omitted to demonstrate the ability to recommend actions without a need for active communications. Although the communications layer was omitted, a fully developed BESS DT would implement an appropriate communication architecture such as OPC UA.

8.4 Key contributions

In the early stages of this work, OPC UA was demonstrated as a means of communication between a National Instruments CompactRIO, an Android app on a smartphone, and a Typhoon HIL simulation. The security-first approach of OPC UA is beneficial to the possible safety-critical nature of DTs, and the flexible information model simplifies interoperability with other devices. Unused networking equipment was repurposed for this demonstration, and the small network created was later expanded to connect more simulators, controllers, and workstations for use in related projects.

To generate DT agent training data, automation tools were created for both Typhoon
HIL and Simulink. These tools specifically included features to launch multiple simulations and record the information needed to create a thorough training dataset. The automation tools were created to facilitate step 3 of the DT agent development process (i.e. run simulations and record data). Although these tools were created for DT development, they can be used to generate data for other ML projects as well.

After a review of related DT literature, the concept of a DT was clarified by identifying virtual twin requirements and using a four-layer architecture to separate the various responsibilities of a DT. Existing work on DTs showed their usefulness for detecting problems, while the BESS DT demonstrates how DT may respond to a problem rather than simply detecting it.

8.5 Conclusion

This work demonstrates how a DT can improve the resilience of nanogrids and microgrids during communication failures. The fundamental idea of a DT is described and an iterative, repeatable DT development process is provided. A DT for a BESS in an industrial nanogrid is created, tested, and evaluated. Evaluation and demonstration of the DT shows that it is capable of recommending actions in response to off-nominal nanogrid events and that it is capable of doing so even while the communication network has failed.
Appendix A

Virtual Twin Model
The virtual twin model is based on the state-space model of the BESS inverter. The schematic of the BESS inverter is shown in Fig. A.1.

Equation A.2 shows the generic form of the state-space model. Equation A.3 shows the forward Euler integration method. Equation A.4 is the virtual twin model derived from a combination of A.2 and A.3.

The virtual twin model is used to calculated the predicted state vector $\hat{x}_{t+1}$ for the next timestep. The difference between the predicted state vector and the measured state vector yields the error vector $\epsilon_t$ as shown in equation A.5.

\[
\frac{d}{dt} x_t = \dot{x}_t = A \cdot x_t + B \cdot u_t \tag{A.2}
\]

\[
\hat{x}_{t+1} = x_t + \Delta t \cdot \frac{d}{dt} x_t \tag{A.3}
\]

\[
\hat{x}_{t+1} = (I + \Delta t \cdot A) \cdot x_t + \Delta t \cdot B \cdot u_t \tag{A.4}
\]

\[
\epsilon_t = \hat{x}_t - x_t \tag{A.5}
\]
The terms in the virtual twin model are: the state vector $x$, the input vector $u$, the state matrix $A$, and the input matrix $B$. Equation A.6 shows the state vector. Equation A.7 shows the input vector. Equation A.8 shows the A matrix. Equation A.9 shows the B matrix.

The state vector $x$, the input vector $u$, and the switch states $S_a$, $S_b$, and $S_c$ in the input matrix $B$ are updated on every timestep $t$. The state matrix $A$ is constant. The values for the state-space model terms are derived from knowledge of the BESS inverter topology.

$$x = [i_{Lfa} \ i_{Lfb} \ i_{Lfc} \ v_{Cfa} \ v_{Cfb} \ v_{Cfc} \ i_{Loa} \ i_{Loc}]^T$$  \hspace{1cm} (A.6)

$$u = [v_{DC} \ v_{DC} \ v_{ga} \ v_{gb} \ v_{gc}]^T$$  \hspace{1cm} (A.7)

$$A = \begin{bmatrix}
-\frac{R_d}{L_f} & 0 & 0 & -\frac{2}{3L_f} & \frac{1}{3L_f} & \frac{R_d}{L_f} & 0 & 0 \\
0 & -\frac{R_d}{L_f} & 0 & \frac{2}{3L_f} & \frac{1}{3L_f} & 0 & \frac{R_d}{L_f} & 0 \\
0 & 0 & -\frac{R_d}{L_f} & \frac{2}{3L_f} & \frac{1}{3L_f} & -\frac{2}{3L_f} & 0 & 0 & R_d & L_f \\
\frac{1}{C_f} & 0 & 0 & 0 & 0 & -\frac{1}{C_f} & 0 & 0 \\
0 & \frac{1}{C_f} & 0 & 0 & 0 & 0 & -\frac{1}{C_f} & 0 & 0 \\
0 & 0 & \frac{1}{C_f} & 0 & 0 & 0 & 0 & -\frac{1}{C_f} & 0 & 0 \\
\frac{R_d}{L_o} & 0 & 0 & \frac{1}{L_o} & 0 & 0 & \frac{-(R_d+R_o)}{L_o} & 0 & 0 \\
0 & \frac{R_d}{L_o} & 0 & 0 & \frac{1}{L_o} & 0 & 0 & \frac{-(R_d+R_o)}{L_o} & 0 & 0 \\
0 & 0 & \frac{R_d}{L_o} & 0 & 0 & \frac{1}{L_o} & 0 & 0 & \frac{-(R_d+R_o)}{L_o} & 0 & 0 \\
\end{bmatrix}$$  \hspace{1cm} (A.8)

$$B = \begin{bmatrix}
\frac{2}{3L_f}S_a & \frac{-1}{3L_f}S_a & \frac{-1}{3L_f}S_c & 0 & 0 & 0 \\
\frac{-1}{3L_f}S_a & \frac{2}{3L_f}S_b & \frac{-1}{3L_f}S_c & 0 & 0 & 0 \\
\frac{-1}{3L_f}S_a & \frac{-1}{3L_f}S_b & \frac{2}{3L_f}S_c & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{-1}{L_o} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{-1}{L_o} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{-1}{L_o} \\
\end{bmatrix}$$  \hspace{1cm} (A.9)
Figure A.2: Measurements and error vectors for A-B fault at generator

Fig. A.2 shows pre-fault and post-fault measurements and error vector components for a line-to-line fault (phase A-B) at the generator, where the vertical line denotes the fault inception time. Fig. A.3 shows measurements and error vector components for a line-to-line (phase A-B) fault at the induction machine. Fig. A.4 shows measurements and error vector components for a three-line-to-ground fault at the induction machine.
Figure A.3: Measurements and error vectors for A-B fault at induction machine

Figure A.4: Measurements and error vectors for A-B-C-Gnd fault at induction machine
Appendix B

Intelligent Agent ML Model
The BESS DT agent contains three neural networks: an encoder, a decoder, and an action recommender. The encoder and decoder are paired to create an autoencoder [152]. The action recommender takes its input from the latent space of the autoencoder. Fig. B.1 shows a diagram of the ML model.

The model uses 21 input channels: 9 state measurements, 9 error vector components, and 3 grid voltage measurements. The encoder outputs parameters (mean and variance) for a Gaussian distribution on the latent space. The latent space is 6-dimensional. A point is sampled from the latent space using the parameters from the encoder. The sampled point is used as input to the decoder and the action recommender. The decoder reconstructs the 9 components of the error vector. The action recommender outputs a one-hot vector corresponding to the recommended action.

The encoder and decoder network architectures contain residual blocks similar to [153]. These blocks consist of two parallel paths. One path contains a convolutional layer followed by an activation layer and another convolutional layer. The other path is simply a fully connected layer. The outputs from these paths are summed together and passed through an activation function. Using the layers in this fashion gives the network the ability to leverage the output from the convolutional layers or to bypass them with the fully connected layer.

![Figure B.1: BESS DT agent ML model](image-url)
The encoder network consists of a sequence input layer followed by several residual blocks and fully connected layers for generating the parameters. Fig. B.2a shows the encoder input layer and several residual blocks. Fig. B.2b shows the output layers of the encoder, including the concatenation of two fully connected layers, one to output the mean and the other to output the variance.

The decoder network operates as the inverse of the encoder. It follows a similar topology as the encoder, but with the layers in reverse and using transposed convolutional layers. Fig. B.3a shows the decoder input layer and the fully connected layers. Fig. B.3b shows several residual blocks and the decoder output.

The action recommender uses residual blocks followed by pooling layers (Fig. B.4a). After the residual layers, a series of fully connected layers is in parallel with a path containing a LSTM layer (Fig. B.4b, LSTM layer in the left path). The outputs of these paths are concatenated and passed through several fully connected layers and a softmax output layer (Fig. B.4c).

Reconstruction plots were created by overlaying the reconstructed components (dots) on the actual values (lines). The top plots show the filter inductor current errors, the middle plots show the filter capacitor voltage errors, and the bottom plots show the output inductor current errors. The autoencoder performed well enough that most of the reconstructions entirely overlay the actual values in the plots. Notable differences in the reconstruction can be seen in the filter inductor current plots.

Fig. B.6 shows reconstructions of a line-to-line (phase A-B) fault at the generator, on the bus, and by the IM. Fig. B.7 shows reconstructions of a three-line-to-ground fault at the generator, on the bus, and by the IM. Fig. B.8 shows the reconstruction of an IM load step.
Figure B.2: Encoder network segments
Figure B.3: Decoder network segments
(a) Recommender input, residual blocks, and pooling layers
(b) Recommender LSTM layer
(c) Recommender output

Figure B.4: Action recommender network segments
Figure B.5: BESS DT agent neural network diagrams
Figure B.6: Reconstructions of A-B fault error vectors

(a) Fault at generator  (b) Fault on bus  (c) Fault at IM

Figure B.7: Reconstructions of A-B-C-Gnd fault error vectors

(a) Fault at generator  (b) Fault on bus  (c) Fault at IM
Figure B.8: Reconstruction of induction machine load step error vectors
Bibliography


[40] D. G. Photovoltaics and E. Storage, “IEEE guide for smart grid interoperability of energy technology and information technology operation with the electric power system (EPS), end-use applications, and loads,” 2011.


