FINAL TECHNICAL REPORT SERC-2022-TR-008

WRT 1049.8.6
HOLISTIC ASSURANCE FRAMEWORK: FAST TIME EMERGENT SCENARIO SIMULATION (FTESS)

Date: July 8, 2022

PRINCIPAL INVESTIGATOR: Dr. Lance Sherry, George Mason University

RESEARCH TEAM

Dr. James Baldo, Dr. Brett Berlin, Dr. Ran Ji, Dr. Ali Raz, Dr. John Shortle, Dr Jie Xu, Jomanah Basatah, Amy Rose (George Mason University)

SPONSOR: OFFICE OF THE UNDER SECRETARY OF DEFENSE FOR ACQUISITION AND SUSTAINMENT

Disclaimer

The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract [HQ0034-19-D-0003, TO#0309].

Any views, opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense nor ASD(R&E).

Research Team

Name	Org.	Labor Category
Dr. Lance Sherry	George Mason University	Principal Investigator
Dr. James Baldo	George Mason University	Faculty
Dr. Brett Berlin	George Mason University	Faculty
Dr. Ran Ji	George Mason University	Faculty
Dr. Ali Raz	George Mason University	Faculty
Dr. John Shortle	George Mason University	Faculty
Dr. Jie Xu	George Mason University	Faculty
Jomanah Basatah	George Mason University	Researcher
Amy Rose	George Mason University	Researcher

Table of Contents

Report Summary

One of the challenges in designing and operating systems composed of interacting components is validating that the emergent behavior of the system does not cause one or more components to migrate, over time, into a hazardous operating state.

Research on several modern airline accidents exhibit the characteristics of Interaction Accidents – no component failed, but the interaction of components resulted in a hazardous state.

Due to the dependence on time, emergent behavior cannot be evaluated by analysis of the design. In theory, it can be evaluated by Digital-Twin agent-based simulations. However, running these simulations to uncover rare event emergent hazardous states is prohibitive due to: (1) the combinatorics of initial states of each of the components, and the (2) combinatorics of the time dimension (i.e. small variations in timing can result in very different outcomes).

Deep Learning Neural Networks (DLNN) have shown promise to capture the underlying combinatoric behavior as well as compress the time dimension.

This report demonstrates the application of DLNN to identify emergent behavior from components with hybrid moded/continuous behavior that plays out over time. DLNNs were trained and tested for three systems with increasing behavioral complexity. The DLNNs accurately were able to represent the time-dependent behavior for which they were trained/tested. The DLNNs were also able to learn and predict emergent behavior for behaviors that were not included in the training/testing data (up to 63% of the missing cases).

These results suggest that DLNN could be used to supplement MBSE/Digital-Twins to increase the operational state-space coverage for System Validation Testing. The implications of these results are discussed.

1 Background: Challenges in System Validation Testing

Increasing technological capabilities have facilitated the development and fielding of systems-of-systems composed of increasingly tightly coupled subsystems and components. In addition to increasingly tightly coupled components, the advent of AI components using Machine Learning has increased the functional complexity of the components. The consequence is a geometric increase in the functional complexity of systems-of-systems increasing their risk of hazardous outcomes and increasing the time, cost, and complexity of testing.

A recent analysis of modern aviation accidents has identified a category of accidents in which no component was considered to have failed (Sherry et.al 2020; Sherry, Mauro, 2019, Sherry, Mauro, 2015). Instead, the sequence of events resulting from the interaction between the components over time resulted in a transition of the system-of-systems (or one of its components) into a hazardous state.

An example of this type of an Interaction Accident is the Singapore Airlines (SQ 237) Runway Excursion at Munich Airport (Sherry, 2019). In this incident, the interaction between 15 systems resulted in a rare event in which an arriving aircraft (SQ 237) left the runway after touchdown (Figure 1).

Example Interaction Accident

There were three main sub-systems: Air Traffic Control, a departing aircraft, and an arriving aircraft. Under pressure to expedite departures that had backed up due to poor weather earlier in the day, the Air Traffic Controller, in compliance with ATC Procedures, authorized a mid-runway departure of the departing aircraft. The aircraft executed the departure in compliance with airline and standard instrument departure procedures. It was heavy, so the climb rate was appropriate by lower than the average departing flight.

Simultaneously, an arriving aircraft was cleared for landing by Air Traffic Control consistent with Air Traffic Control procedures. The timing of the arriving and departing aircraft was such that the departing aircraft would have cleared the runway in advance of the arriving aircraft performing a landing.

Consistent with airline Standard Operating Procedures, the flight crew chose to perform an Autoland Category III approach. The weather conditions did not require a Category III approach, but the flight crew elected to use this opportunity to practice the Category III procedure and maintain proficiency. This choice was also consistent with airline policies and procedures.

The arriving and departing aircraft were *coupled* by an Instrument Landing System (ILS). The ILS Localizer signal was deflected by the slow climbing departing aircraft as the arriving aircraft was about to Land causing the arriving aircraft Autoland system to bank the aircraft resulting in a landing that steered the aircraft adjacent to the runway. The timing of the deflection could not have been worse.

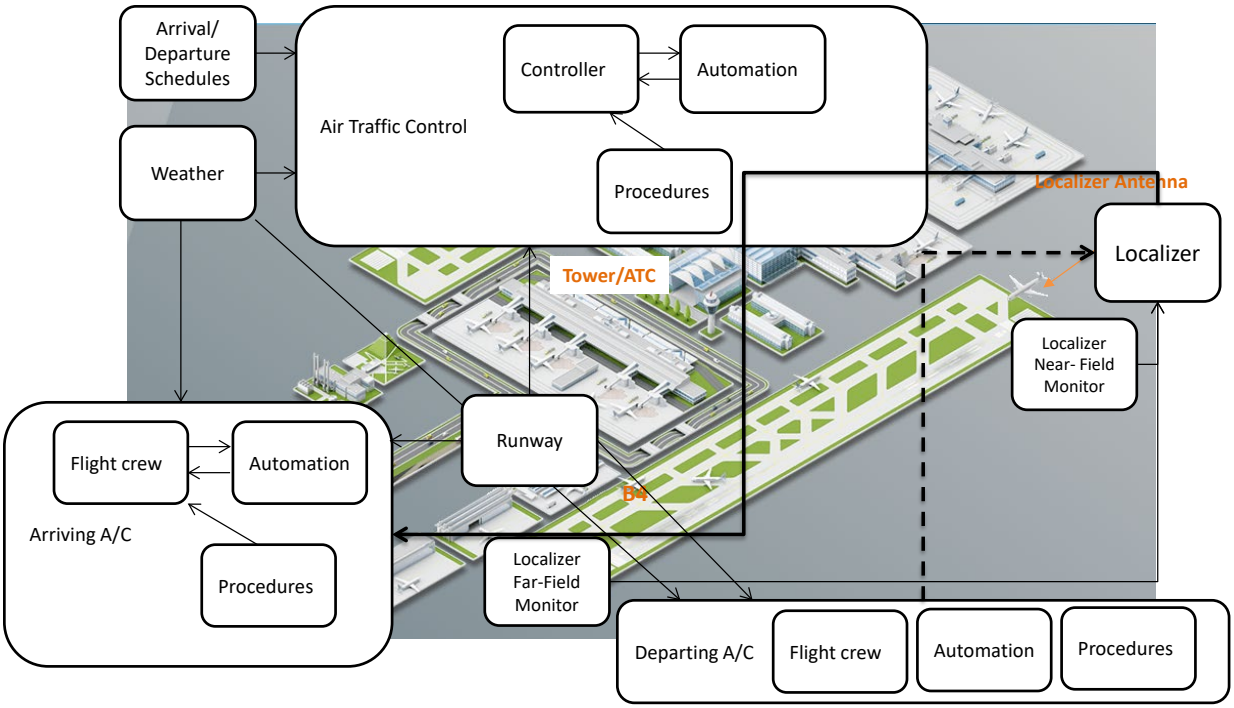


FIGURE 1: Fifteen components interact. The accident was precipitated by a tight coupling between a departing aircraft, the Localizer, and the arriving aircraft Autoland at just the wrong moment.

Attempts to replicate the accident in a Digital-Twin agent-based model (Figure 1) have been stymied by the combinatorial complexity of 15 component systems (Sherry, Mauro, 2019). Running the simulation to cover the combinatorics of initial conditions (even with a Super Computer) is prohibitive. Further, the time dependence of emergent behavior adds to the complexity (i.e. small variations in the timing result in different emergent outcomes).

This phenomenon was first described by Charles Perrow as “Normal Accidents” (Perrow, 1984) and has more recently been labeled as an Interaction Accident (IA) or a Functional Complexity Failure (FCF) to capture the source of the failure (i.e. functional complexity).

Interaction Accidents are very difficult to prevent by design and/or by testing for four reasons.

1. they require the evaluation of a prohibitively large number of combinations of events for rare hazardous outcomes.
2. they require seamless and transparent coordination of the interfaces between a large number of tightly coupled components.
3. the seamless and transparent integration must occur through disparate organizations in the supply chain including those responsible for maintenance of the component after its fielded.

4. in almost all of the cases, the hazardous state was precipitated by the result of a seemingly innocuous change to a peripheral component long after the system was tested, certified, and fielded.

How is System Validation Testing of Systems-of-Systems Done Today?

Systems-of-systems are designed largely by manual methods that rely on the *imagination of the designers* to identify sequences of events that can lead to hazardous outcomes. The design process has been significantly enhanced by Model-based System Engineering (MBSE) tools that enable the creation of models, simulations, and digital-twins. When the MBSE models are executable, the digital twin can be simulated. However, the functional behavior of the components, and the system architecture resulting in the coupling of components, are the result of the intellectual labor of the designers. As a consequence, the Accident reports for these accidents always cite the "lack of imagination" of the designers.

In theory, the MBSE models/simulations/digital-twins provide the means to exhaustively test the interaction of the components over time. However, these models are subject to issues:

1. the complexity of the behavior of the individual components
2. the fidelity of the behavior of the components (i.e. matching the actual behavior of fielded components
3. the combinatorics of the interactions between the components
4. the combinatorics of the interaction between the components over time

Even for moderately behaviorally complex components with moderate coupling, the combinatorics of the interactions over time make the simulation of the complete operational state-space time and cost-prohibitive.

To use the digital-twin to address Functional Complexity Failures (FCFs) in systems-of-systems requires three functions: (1) a function for integrating the models of the tightly coupled systems-of-systems components from multiple vendors in the supply chain, (2) a function for "combinatoric seeding" of scenarios within a Monte Carlo shell, and (3) a function for post-processing the simulation results to identify hazardous outcomes.

The Need for Improved System Validation Testing of Systems-of-Systems

DoD Acquisitions must ensure that the design/testing (i.e. digital twin) and operational maintenance of fielded systems-of-systems composed of tightly coupled AI components, meets system-of-systems requirements with regard to the Target Level of Safety (TLS) and the risk of hazardous outcomes. In addition to system performance considerations, there are also budget and schedule constraints that can limit the quality and quantity of testing that can be conducted.

Concept-of-Operations: Fast-Time Emergent Scenario Simulations (FTESS)

Fast-Time Emergent Scenario Simulation (FTESS) is an approach to reducing the risk of hazardous outcomes from tightly coupled systems-of-systems (Sherry et.al. 2019).

FTESS is an agent-based, rare-event simulation “digital-twin” of the system. The components of the system-of-systems are modeled by mid-fidelity simulations.

The FTESS is run using Monte Carlo techniques (Nanduri & Sherry, 2017) on a super-computer or other parallel computing platforms (Snisarevska, Sherry et.al., 2018). Inexpensive access to cloud-based Supercomputing. GMU has access to the Argos Supercomputing Cluster

An important feature of the FTESS is its ability to generate rare events. FTESS leverages techniques for rare-event simulation, edge computing, and runs on a supercomputer. There are two main approaches for improving the efficiency of rare-event simulations— Importance Sampling (IS) and Splitting (Shortle and L’Ecquier, 2011; Zare-Noghabi & Shortle, 2017).

The idea of IS is to use statistical sampling methods to identify the combinatorics for the simulations most likely to yield hazardous outcomes. IS is also useful for systems that tend to take a small number of ‘‘catastrophic jumps’’ to the rare event.

The idea of Splitting is to create separate copies of the simulation whenever the simulation gets ‘‘close’’ to the rare event of interest, effectively multiplying promising runs that are more likely to reach the rare event. Splitting is useful for systems that tend to take many incremental steps on the path to the rare event.

FTESS is run during the development and testing cycle to identify emergent scenarios leading to hazardous outcomes such that they can be addressed by the design. More Importantly, FTESS is run even when the system of systems is fielded and operating. This can be used to identify hazardous outcomes that have not yet occurred.

2 Concept of Operations: Fast Time Emergent Scenario Simulation (FTESS)

Fast-Time Emergent Scenario Simulation (FTESS) is an approach to reducing the risk of hazardous outcomes from tightly coupled systems-of-systems (Sherry et.al. 2019). FTESS is an agent-based, rare-event simulation “digital-twin” of the system. The components of the system-of-systems are modeled by mid-fidelity simulations.

The FTESS is intended to support the Concept Definition, Development, Production, and Operational Phases of the system life-cycle. The FTESS increases the slope of the learning-curve during the Concept Definition and Development phases of the project. The FTESS provides the

ability to conduct feasibility studies, fine-tune and demonstrate Concept-of-Operations, and establish thresholds and parameters for design requirements.

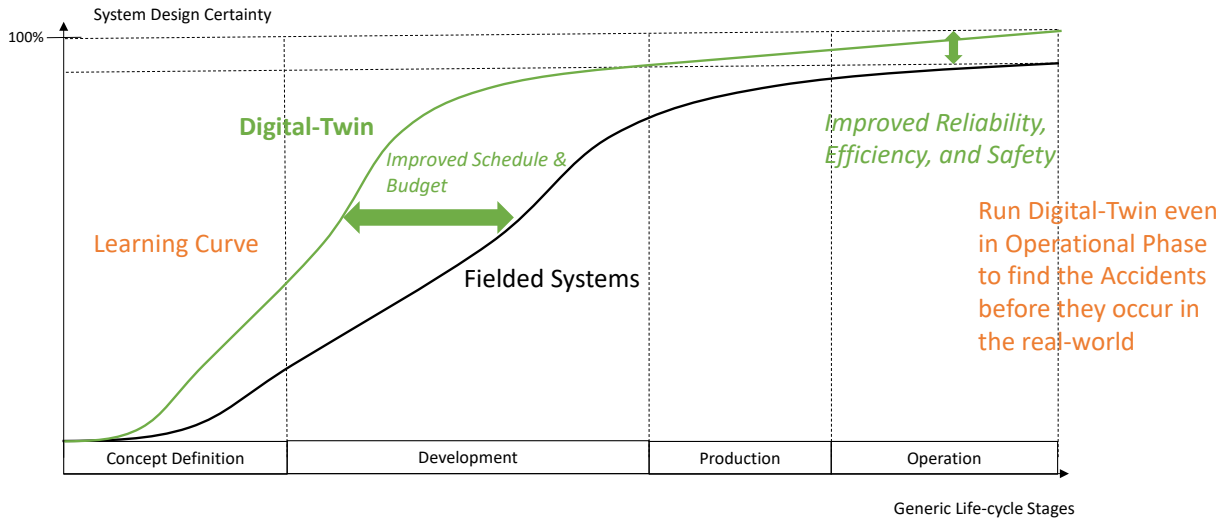


FIGURE 2: Digital-Twin/MBSE simulations can increase the slope of the learning-curves resulting in an improved schedule and budget during the Concept Definition and Development phases. Digital-Twin/MBSE simulations can also be applied in the Production and Operations phases to improve reliability, efficiency, and safety.

The FTESS is also applicable to the Production and Operations phase. During these phases, the FTESS is operated in parallel with actual operations. The idea here is to identify the potential for hazardous events in the digital-twin simulation before they occur in the real-world (Figure 3).

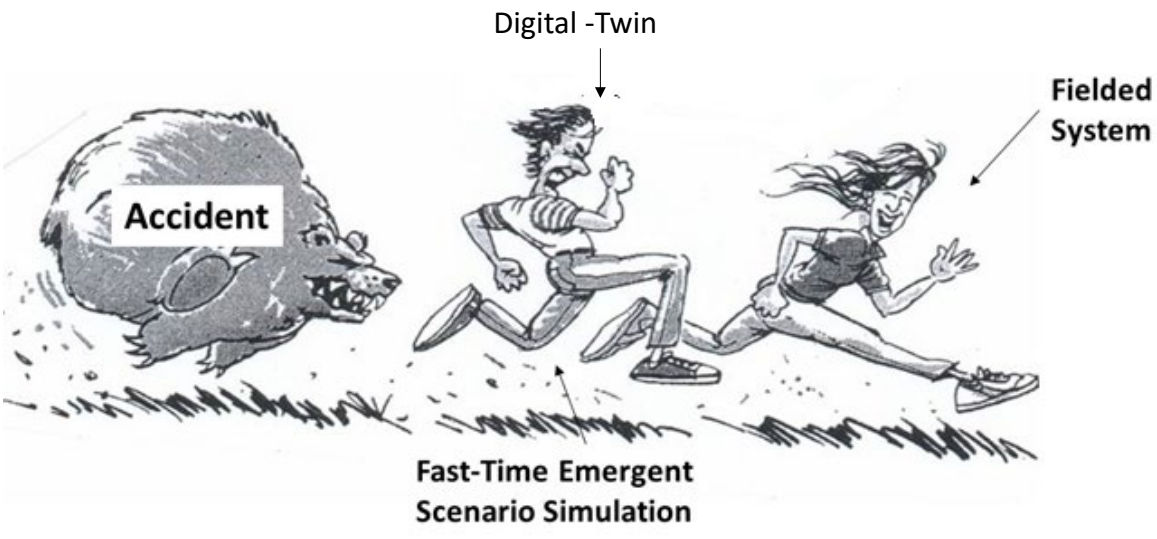


FIGURE 3: FTESS concept is to identify the potential for hazardous events in the digital-twin simulation before they occur in the real-world

Terminology – Tightly Coupled System-of-Systems

A System is composed of tightly coupled, interacting components (Figure 4). The components exhibit complex behaviors generated by a hybrid model/logical and continuous algorithms. These components may be ML-derived AI, and may exhibit adaptive behavior. For the purpose of this report, exogenous inputs from Environment are captured as a component of the System.

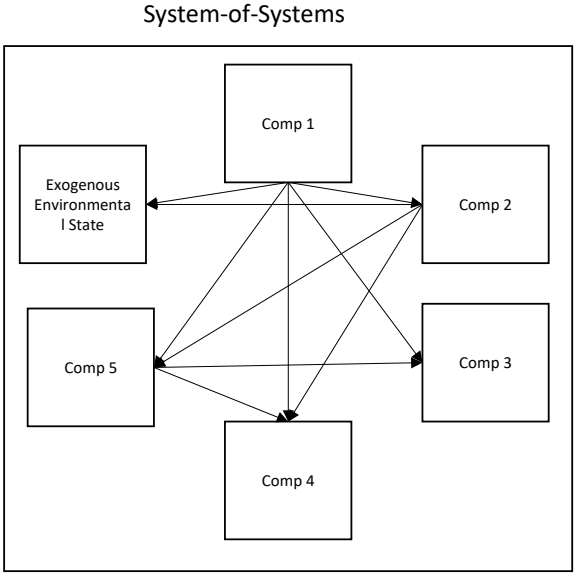


FIGURE 4: System is composed of coupled components. Components exhibit complex behaviors associated with hybrid moded-logic and continuous algorithms.

Overall System performance is measured by parameter $C_i x_j(t)$, where C_i is for component I, and $x_j(t)$ parameter j as a function of time. The parameter $C_i x_j(t)$ varies over time (Figure 5). Each parameter has an associated safe operational regime (e.g. max/min speed).

Overall system performance is determined *over time* based on a set of Initial Conditions $C_i x_j(0)$. The value of the parameter at any given time is a function of the interactions between components. The parameter trajectory is based on the behavior of the components, as well as the *timing of the interactions*.

The *operational state-space* of the System is defined by the outcomes of the interactions for every combination of Initial Conditions. The combinatorics of the operational state-space is a function of the combinatorics of the Initial Conditions *as well as* the timing of the interactions.

When Systems behavior is bounded by components reset (e.g. end-of-day, or exit runway), or when Systems have periodic repeatable behavior, the System behavior can be defined by the pair:

- Initial Conditions - $C_i x_j(0)$ for all i and j.
- Final State - - $C_i x_j(t^*)$ for all i and j.

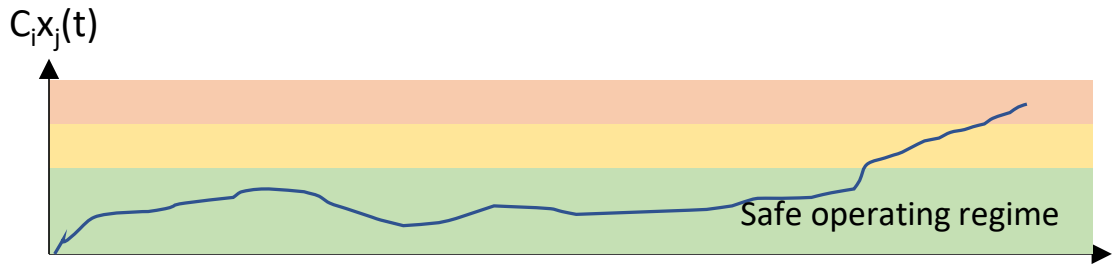


FIGURE 5: Parameter for a Component varies over time. Parameter has an associated safe and hazardous operating regimes.

Traditional Model-based Simulation/Digital-Twin for System Validation Testing

In modern, traditional System Engineering practice, Con-Ops, Requirements, Design, Verification Testing, and Validation Testing documentation are developed through a manual process. The process inherently relies on the imagination of the design engineers.

Advances in engineering practice have leveraged the capabilities of simulation to support the intellectual labor of design using Model-based/Digital-Twin simulations to explore the operational state-space of designs (Figure 6).

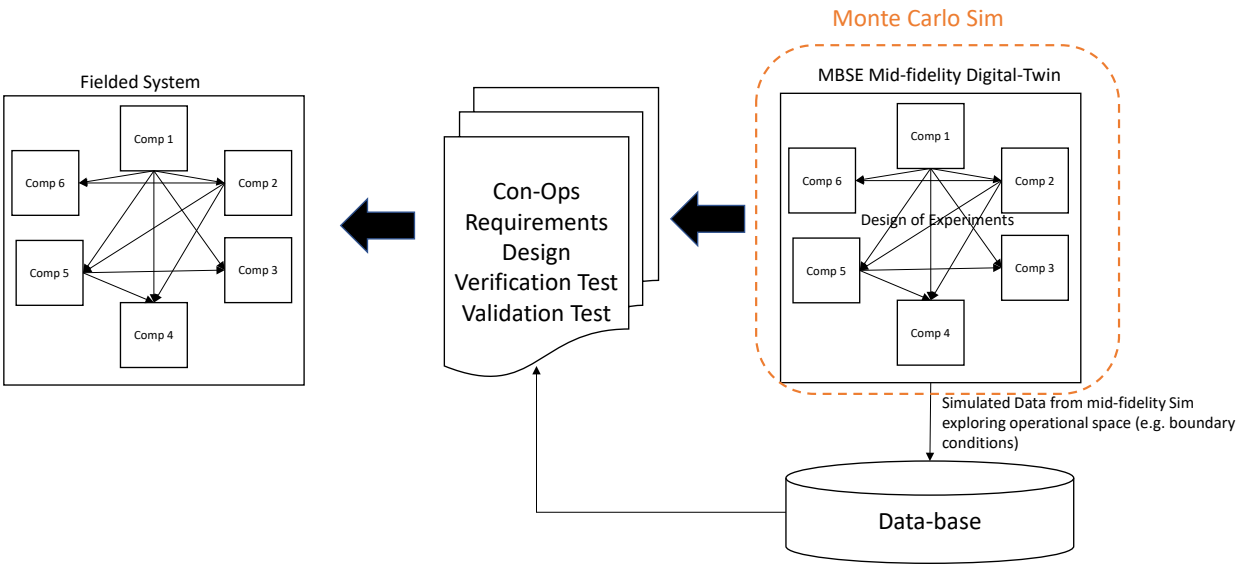


FIGURE 6: Advances in engineering practice have leveraged the capabilities of simulation to support the intellectual labor of design using Model-based/Digital-Twin simulations to explore the operational state-space of designs

Holistic Assurance Framework: Fast Time Emergent Scenario Simulation (FTESS)

MBSE/Digital-Twin simulations have demonstrated the ability to significantly enhance the intellectual labor of the designers. These simulations, however, exhibit some limitations:

1) The combinatorics of the operational state-space is dependent on the combinatorics of the Initial Conditions *as well as* the combinatorics of the timing of interactions. Even using super-computers and advanced processors, the combinatorics of the interactions can be time-prohibitive for Model-based/Digital-Twin simulations to cover the complete operational state-space.
2) simulations may not capture corner cases due to missing some $C_i x_j(t)$
3) missing transient dynamics. Note: steady-state dynamics are generally very good.

Researchers (on this team) have developed approaches for improving the efficiency of rare-event simulation: (1) Importance Sampling (IS), and (2) Splitting (Shortle and L'Ecquier, 2011; Zare-Noghabi & Shortle, 2017). The concept behind Importance Sampling is to use statistical sampling methods to identify the combinatorics most likely to yield hazardous outcomes. The computational processing time is then used to simulate these combinatorics.

The concept behind Splitting is to create separate copies of the simulation whenever the simulation gets ''close'' to the rare event of interest, effectively multiplying promising runs that

are more likely to reach the rare event. Splitting is useful for systems that tend to take many incremental steps on the path to the rare event.

Another approach to overcome the limitations, a DLNN is proposed to supplement the MBSE/Digital-Twin simulations (Figure 7). The DLNN, trained on operational and/or simulation data, learns the underlying behavior of the System. The DLNN can then be used to extend the operational state space by generating scenarios, not in the original DLNN Training/Testing data.

The next section demonstrates the application of DLNN to perform System Validation Testing.

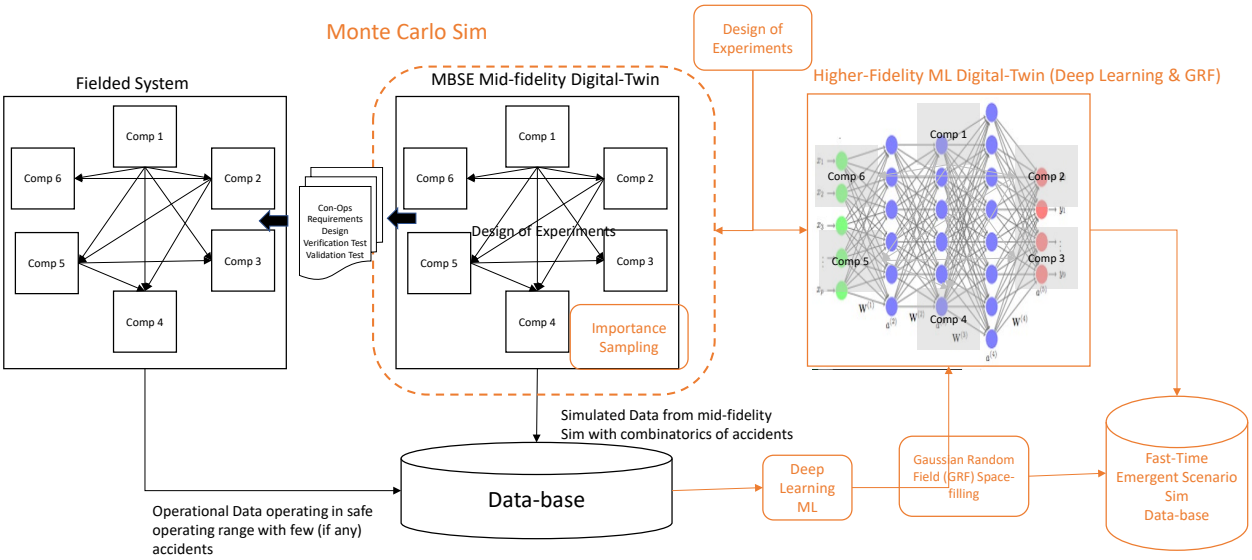


FIGURE 7: Application of Deep Learning Neural Network (DLNN) to supplement the MBSE/Digital-Twin Support of System Validation. The DLNN learns the underlying behavior of the System and can generate Initial Condition/Final State pairs for scenarios in the operational state-space not covered by the MBSE/Digital-Twin simulation.

3 Demonstrating the Use of Deep Learning Neural Network for Digital-Twin System Validation Testing

3.1 Introduction

Analysis of modern airline accidents has identified that many of these accidents are **not** the result of one or more component failures. Instead, these accidents are the result of *complex interactions* between the components in a systems-of-systems that migrated the system into a hazardous operational regime.

This class of accidents are known as Normal Accidents (Perrow, 1984), Functional Complexity Failures (Sherry, Mauro, 2015), or System Component Interaction Accidents (Sherry, Mauro, 2019).

An example of this type of System Component Interaction Accident (SCIA) is the Singapore Airlines (SQ 237) Runway Excursion at Munich Airport (Sherry, 2019). In this incident, the interaction between 16 systems resulted in a rare event in which an arriving aircraft (SQ 237) left the runway after touchdown. The accident was precipitated by a tight coupling between a departing aircraft, the Localizer, and the arriving aircraft Autoland.

Attempts to replicate the accident in a Digital-Twin agent-based model (Figure 1 above) have been stymied by the combinatorial complexity of 15 component systems (Sherry, Mauro, 2019). Running the simulation to cover the combinatorics of initial conditions (even with a Super Computer) is prohibitive. Further, the time dependence of emergent behavior adds to the complexity (i.e. small variations in the timing result in different emergent outcomes).

The Promise of Deep Learning Neural Networks

Deep Learning Neural Networks (DLNNs) have shown promise for capturing complex behavior. DLNNs are designed for high-quality pattern recognition (McCulloch & Pitts 1943; Rosenblatt, 1985). They have been successfully applied to a wide variety of pattern recognition problems in science and industry, including mastering the game of Go (Silver et al 2016). For example, the famous Newton 3-body problem has been successfully modeled by a DLNN (Breen et.al. 2020). Modeling this system is considered intractable (see de Lagrange, 1775). Currently, the solution for a given initialization can only be found by performing laborious iterative calculations that have unpredictable and potentially infinite computational costs. The DLNN, "over a bounded time interval, provides accurate solutions at a fixed computational cost and up to 100 million times faster than the numerical integrator." (Breen et.al. 2020).

DLNNs capability is a function of two properties:

(i) closely approximating any continuous function that describes the relationship between an outcome and a set of covariates, known as the universal approximation theorem (Hornik 1991; Cybenko 1989)

(ii) once trained, a DLNN has a predictable and a fixed computational burden (i.e. it behaves like a "look-up table.").

The example applications of DLNN are for pattern recognition in an image, decision making, and continuous functions. Berlin et. al. (2022) have demonstrated DLNN for hybrid moded/continuous systems.

This report demonstrates the results of training/developing DLNNs for systems with:

1) Combinatorial complexity due to the large number of combinations of initial conditions

2) Behavioral complexity due to hybrid moded/continuous functional behavior

3) Combinatorial complexity due to time dependence of interactions

4) Predicting emergent behavior beyond the range of the data used for training/testing the DLNN

The fourth requirement - predicting beyond the range of training data - is important. Due to the limitation of not being able to run the simulation for all the combinations of initial conditions and timing variations, it is not possible to use a complete set of data to train the DLNN. The DLNN must "learn" the underlying behavior and be able to predict emergent outcomes that were not part of the training data.

This section describes the outcome of an experiment to determine the accuracy of DLNNs trained for three incrementally more complex systems with hybrid moded/continuous behaviors. The experiment also evaluated the DLNNS capability to compress the time dimension effectively converting the time-dependent behavior into a "look-up table."

3.2 The Goal: Train DLNN to Predict Outcomes Not in Original Training Data Set

The goal of this experiment is to demonstrate the capability of DLNN to learn the behavior of a System of interacting components, and then be able to predict interaction outcomes over time for System initial states that were not in the DLNN Training/Testing data set (Figure 8).

The concept is as follows: There exists a theoretical list of the Initial Conditions/Final State pairs for a System of interacting components (shown on the left of Figure 8). A simulation/digital-twin is developed and used to generate Initial Conditions/Final State pairs (second column from left in Figure 8). This list is a subset of the theoretical complete list due to limitations of simulation processing time and cost.

The Digital-Twin DLNN is developed using this subset of data. First a Hold Out data is set is spilt from the simulation data (shown in column 5 of Figure 8). The remaining data is used to training/test the DLNN using a randomized split between training/testing data. The data set is also used for k-fold cross validation testing to adjust the hyper-parameters in the DLNN training/testing (shown in column 4 of Figure 8).

The DLNN is then tested against the Hold Out Data to evaluate how well the DLNN learned the underlying behavior of the System of interacting components(shown in column 5 of Figure 8).

Finally the DLNN is used as "look-up table" to predict the final state outcomes for Initial Conditions not generated by the simulation/digital-twin.

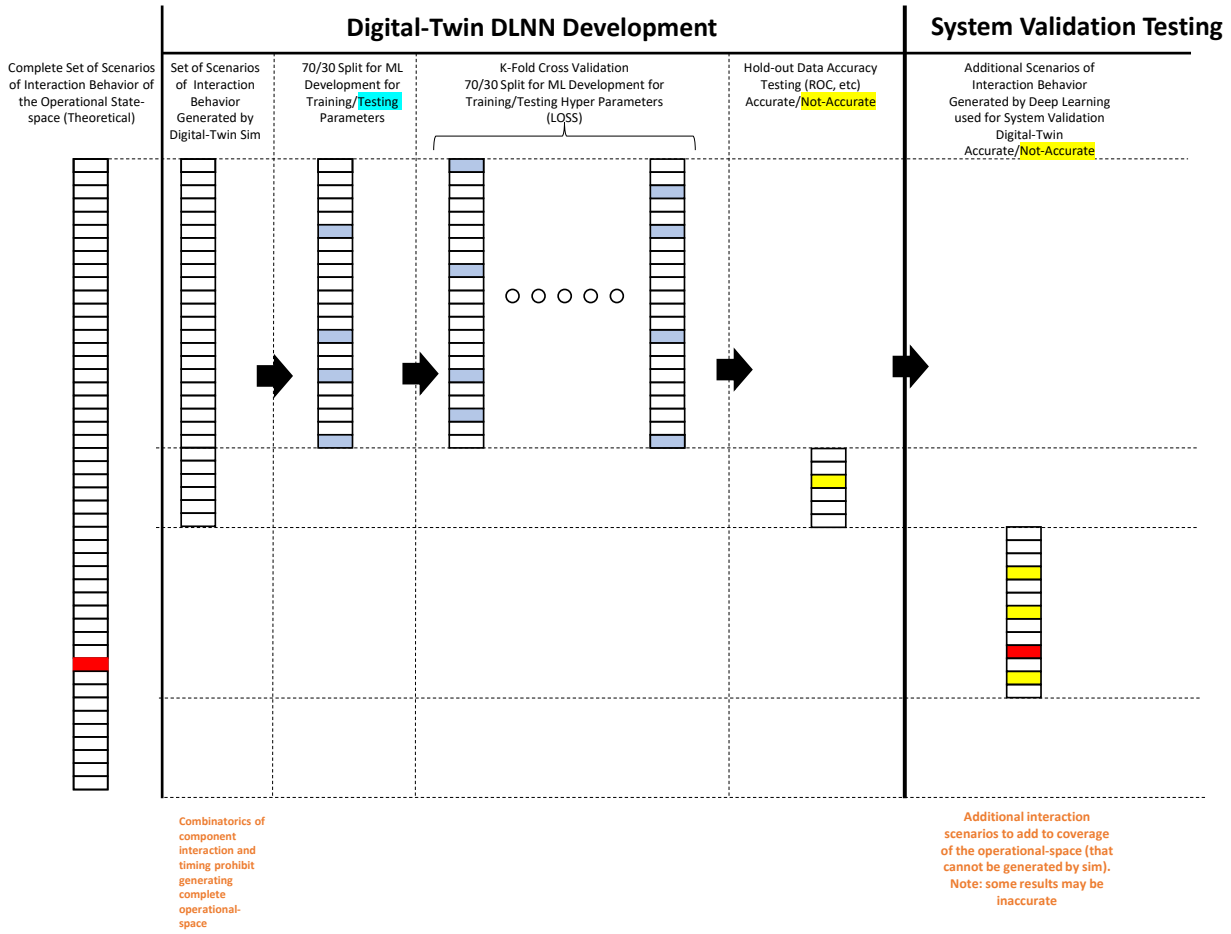


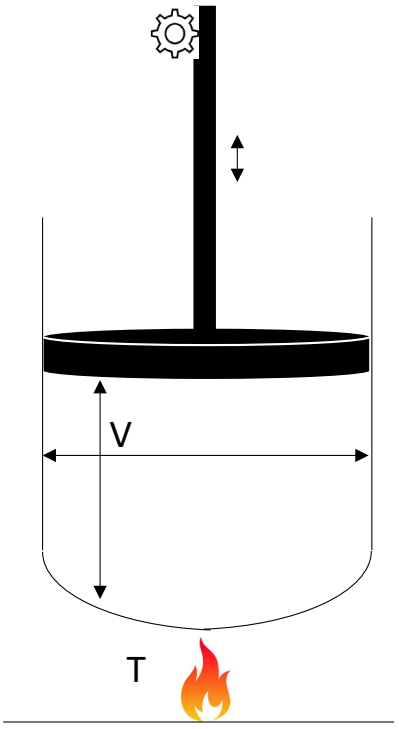
FIGURE 8: The concept of using simulation data to train a DLNN on the behavior of a System of interacting component. The DLNN, learns the underlying behavior, and then is able to predict System outcomes over time for initial conditions not in the original simulation data set.

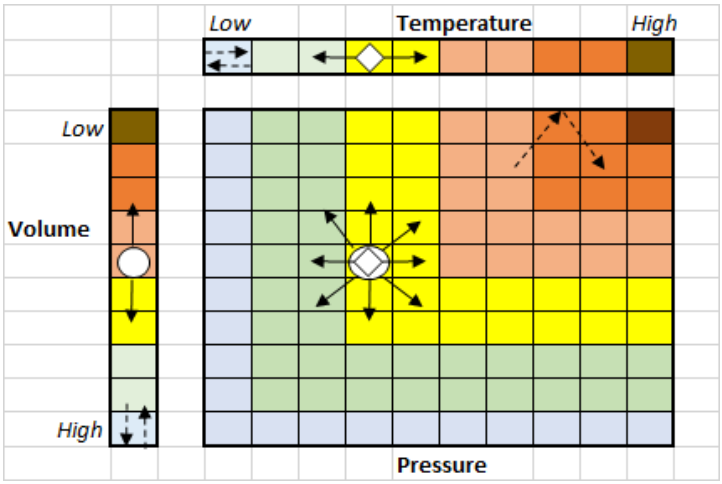
This section is organized as follows. Section 2 describes the three systems of increasing complexity to be evaluated. Section 3 describes the development training/testing of the DLNN, and the testing of the "out of range" data. Section 4 describes the results of the development training/testing of the DLNN, and the testing of the "out of range" data for each of the three systems.

3.3 Example System for Study of DLNN Capture of Complex Interaction Behavior

A hypothetical system is postulated with two compressible vessels filled with a hazardous gas. The gas in each vessel follows the Ideal Gas law (i.e. PV=nrT). The vessels are located in a plant such that their Temperature (T) is determined by the adjacent equipment and/or the environment. The vessel's Volume (V) is also determined by the actions of the plant. The vessel

structure is designed to be within the Pressure (P) determined by the expected range of T and V. The components of a single vessel are illustrated in Figure 9.





$$P = nRT/V$$

FIGURE 9: Single vessel Pressure (P) is determined by Temperature (T) and Volume (V) that are determined by the actions of the plant (i.e. not regulated). The state of the vessel T and V can be visualized in arrays and matrix on the right.

The states P, V, and T can be visualized by the charts on the right in Figure 9. Vessel V, determined by the plant actions on the plunger is shown by the vertical array. V increases and decreases until it reaches the upper and lower bounds, at which time it reverses direction.

Vessel T, also determined by the plant actions, is shown on the horizontal array. T increases and decreases until it reaches the upper and lower bounds, at which time it reverses direction.

The combined effect of T and V, P, is shown by the color code in the matrix. The upper right is the highest P. The lower left is the lowest P.

When an individual vessel's volume is compressed in conjunction with an increase in temperature of the gas in the vessel, the Pressure in the vessel may exceed the structural limits of the vessel and result in a structural failure and release of the gas. This constitutes a hazardous event that must be avoided.

The vessels are located and operated in the plant with a sufficient distance such that the Temperature of each vessel does not affect the Temperature of the other vessel (i.e. the P, T, V state of each vessel is not coupled to the other vessel.

However, several years after installation, to accommodate un-related changes in the plant, the vessels are moved in close proximity such that the Temperature of one vessel affects the Temperature of the other vessel. In this way, the state of the Vessels 1 and 2 become coupled.

The coupling results in an increase in the ambient temperature and shift in the T range of Vessels 1 and 2 (see Figure 10). Under specific circumstances, the P in either of the Vessels can now reach a Breach Pressure at which the Vessel can fail.

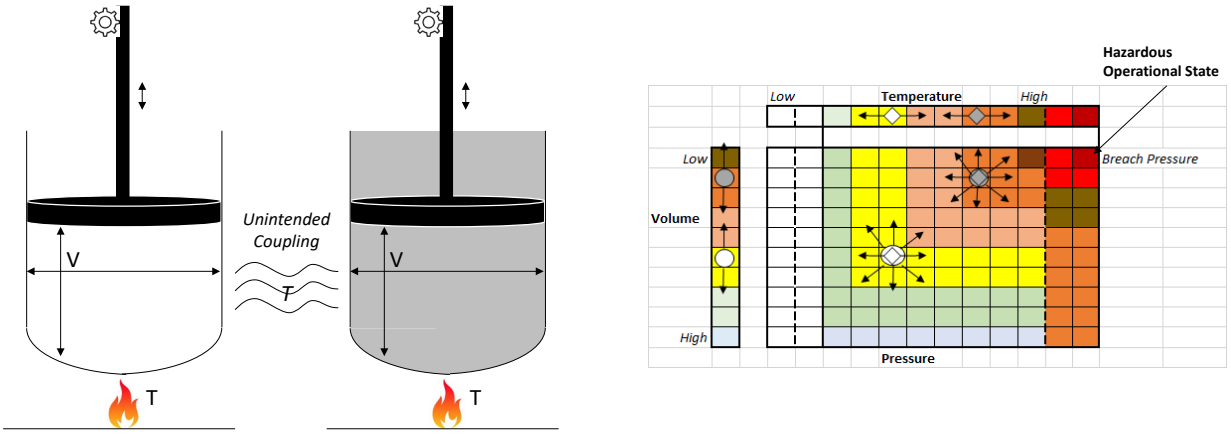


FIGURE 10: Vessels in close proximity are coupled resulting in an increase in the ambient temperature and shift in the range of each Vessel T. The vessel P can now reach a Breach Pressure resulting in a hazardous outcome.

3.4 Experiment: DLNN for System Validation Testing

This section describes the experiment conducted. DLNN models are developed and tested for accuracy in capturing the complex moded/continuous behavior of interacting systems.

A. Definition of System Behavior to be Tested

The experiment described in this paper will evaluate the capability of DLNN to capture the behavior of the three system behaviors shown in Table 1.

TABLE 1: Three system behavior to be tested in the DLNN Experiment

Behavior Dimensions	Component from Hypothetical Example Above	Coupling

One	Temperature (or Volume)	None
Two	Pressure = *f*(Temp, Vol)	Temp coupled to Vol
Four	Pressure = *f*(Temp, Vol)	Two times Temp coupled to Vol

For each component, the DLNN will be developed (i,e, trained and tested) with: (1) the full complement of data per traditional NN training/tests approaches (e.g. 70%/30% split), and (2) an 80% complement of data that is split 70%/30% for development training/testing. The remaining 20% is used for testing and provides a measure of the ability of the DLNN to learn the underlying behavior of the system and be able to predict the outcomes that were not part of the training/testing set.

The accuracy of the DLNN is therefore tested as described in Table 2.

TABLE 2: DLNN Accuracy Tests

Development Training/Testing	**Accuracy Method**
Full complement data	Test based on the full complement of data
Partial Complement (withholding data for testing)	Test based on "hidden data"

B. Deep Learning Neural Network

This experiment in the application of DLNN for the purpose of System Validation was conducted using Momentum AI (Accure, Inc.). Momentum is a data engineering and AI/ML platform that automates various steps of AI/ML development. It provides no-coding toolkits to rapidly train and deploy ML models in production (Figure 11). Momentum provides the following functions:

1. Data engineering, including ingestion, transformation, and automated data pipeline.

2. AI/ML model training using GUI drag-and-drop mechanism for selection of ML algorithms, model, configuration, dataset input, and monitoring of training performance.

3. AI/ML model testing, evaluation, and deployment in production on MLOps, including versioning and feature store maintenance.

4. Monitoring of model usage, performance, data drift detection, error monitoring, anomaly detection, and reporting.

5. Automation of model retraining

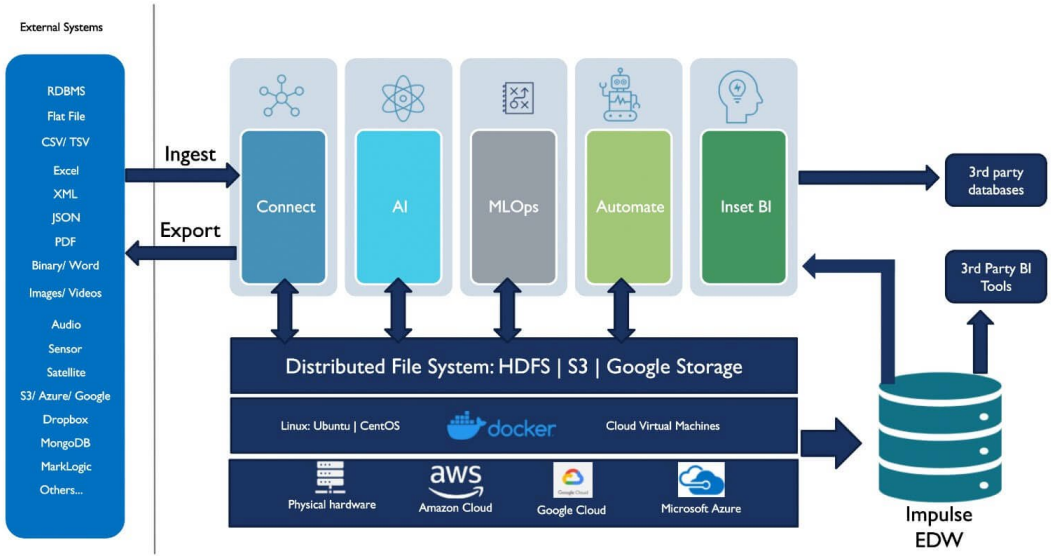


FIGURE 11: Momentum AI provides an end-to-end low code way to create DLNN

For the experiment, Momentum's Multilayer Perceptron Classifier with supervised learning is used. The following characteristics are defined:

1. Feature fields (i.e. Inputs to the DLNN model, such as the Initial States for the systems described above)

2. The initial states are considered categorical variables for the experiment.

3. The categorical variables are encoded using One-Hot encoding algorithm.

4. Target variable field(s) (i.e. Outputs from the DLNN model such as the Final States for the systems described above)

5. Number of possible outcome states

C. Training/Testing Data

The data for the training was generated by a simulation of each of the three systems coded in Visual Basic.

The One Dimension model (e.g. Temp or Vol) has 10 states (zero through nine). The system is initialized with a State and a Direction (i.e. increasing or decreasing) and runs for 10 time steps. When the system state reaches Zero or Nine (i.e. boundary), the mode reverses direction. For

this version of the experiment, the rate of change of the state is constant (i.e. deterministic, not stochastic).

The Two Dimension model (e.g. Pressure = f(Temp, Vol)) has 100 states (zero through ninety-nine). The system is initialized with a State and a Direction (i.e. 8 cardinal directions) and runs for 20 time steps. When the system state reaches any boundary, the mode reverses direction by the angle of incidence is equal to the angle of reflection. For this version of the experiment, the rate of change of the state is constant (i.e. deterministic, not stochastic).

The Four Dimension model has two Two Dimension models running simultaneously. The simulation runs until the Pressure of either vessel (i.e. combination of T and V) exceeds a predefined Pressure Breach limit (i.e. upper right) or until 20 time steps are reached.

Note, that these systems exhibit repeating (i.e. duplicate) behavior and can be halted at an upper time limit without losing any behavior. This may not be the case for all systems.

3.5 Results

This section describes the results of the six experiments conducted: system behaviors (3) X development training/testing and accuracy checks (2).

A. Full Complement DLNN Development Training/Testing with Full Complement Data Accuracy Testing

DLNNs were developed (i.e. trained and tested) with the full complement of data generated by the simulation described above as shown in Table 3. For the purpose of this experiment, the Four Dimension model was constrained to only four initial states for each Vessel (i.e. 16 combinations of initial states).

The DLNN was able to replicate the system behavior 100% for each of the three models. These results were repeatable through 10 development replications.

TABLE 3: Results for Full Complement DLNN Development Training/Testing with Full Complement Data Accuracy Testing

Experiment Characteristics and Results	**One Dimension (e.g. Temp or Vol)**	**Two Dimension (e.g. Pressure = *f*(Temp, Vol)**	**Four Dimensions (e.g. Pressure for two coupled systems)**
Data Set for Development Training/Testing (70/30)	20	800	1024
Hidden Layers	3	5	3
Nodes	64	64	64
Epochs	100	500	500

Output Layer	1	1	1
Inner Layers	2	3	6
Feature Fields	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol) for Vessel 1 and Vessel 2
Non-numeric/Categorical Feature Fields	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol) for Vessel 1 and Vessel 2
OneHotEncode Fields	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol) for Vessel 1 and Vessel 2
Target Field	Final State after 10 time steps (e.g. Temp, Vol)	Final State after 10 time steps (e.g. Pressure)	Final States after 20 time steps (e.g. Pressure 1 and Pressure 2)
Number of Classes	2	2	2
Model Development Accuracy	100%	100%	100%

B. Partial Complement DLNN Development Training/Testing with "Hidden" Data Accuracy Testing

DLNNs were developed (i.e. trained and tested) with a partial complement (i.e. 80%) of behavior data generated by the simulation described above as shown in Table 4. For the purpose of this experiment, the Four Dimension model was constrained to only four initial states for each Vessel (i.e. 16 combinations of initial states).

The DLNN development training/testing was able to replicate the system behavior 100% for each of the three models. These results were repeatable through 10 development replications.

The DLNN model accuracy using "hidden data" (i.e. data that was not used during development training/testing) exhibited a range of performance μ = 100%, 98%, and 68.3% for the One Dimension, Two Dimension, and Four Dimension models respectively, σ = 1.2%, 1.7%, and 9.8%. The range of performance was the result of "hiding" different data sets. For example, when data sets related to change in mode (i.e. direction) were hidden, the DLNN prediction performance was degraded.

TABLE 4: Results for partial complement DLNN Development Training/Testing with Full Complement Data Accuracy Testing

Experiment Characteristics and Results	**One Dimension (e.g. Temp or Vol)**	**Two Dimension (e.g. Pressure = *f*(Temp, Vol)**	**Four Dimensions (e.g. Pressure for two coupled systems)**
Data Set for Development	17 out of 20	640 out of 800	820 out of 1024

Training/Testing (70/30)			
Hidden Layers	3	5	3
Nodes	64	64	64
Epochs	500	500	500
Output Layer	1	2	1
Inner Layers	2	3	6
Feature Fields	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol) and Direction (i.e. 0 to 7) for Vessel 1 and Vessel 2
Non-numeric/Categorical Feature Fields	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol) and Direction (i.e. 0 to 7) for Vessel 1 and Vessel 2
OneHotEncode Fields	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol)	Initial State (e.g. Temp, Vol) and Direction (i.e. 0 to 7) for Vessel 1 and Vessel 2
Target Field	Final State after 10 time steps (e.g. Temp, Vol)	Final State after 10 time steps (e.g. Pressure)	Final States after 20 time steps (e.g. Pressure 1 and Pressure 2)
Number of Classes	9	2	2
Model Development Accuracy	100%	100%	99.8%
Model Testing Accuracy with "Hidden" Data	μ 98.9% σ = 1.2%	μ = 83.5% σ = 3.7%	μ = 68.3% σ = 9.8%

4 Conclusion and Future Work

The results of the experiment show the potential for using DLNNs to overcome limitations in System Validation Testing of the complex, time-dependent behavior resulting from the interaction of components over time.

DLNNs have exhibited the property of being able to "learn" the underlying behavior of a system using only a portion of the complete set of system behavior trajectories and correctly predict outcomes that were not explicitly in the training/testing data. This property is extremely important for System Component Interaction Accident (SCIA) in which all the possible interaction trajectories are not known *a priori*. Although the testing for "hidden" data performance was ~70%, this is better than "0%" which is the current standard. Using DLNNs can successfully evaluate the behavior outside the range of the DLNN training data.

This report describes the results of an experiment to develop DLNNs for three classes of systems with incrementally increasing behavioral complexity. The main results are:

(1) the DLNN was successfully trained for all three classes of behaviors using the full data set with close to 100% accuracy. This performance was sustained across multiple development repetitions

(2) the DLNN was successfully trained for all three classes of behaviors using the partial data set with close to 100% accuracy. However, the testing accuracy for the "hidden data" not used in development degraded to ~70% as the behavioral complexity increased. Further, the performance varied for multiple development and testing repetitions. The reason for the variance is thought to be due to the types of "hidden data" not used for training. Such as data that includes mode changes.

Future work is proposed to continue exploring this concept with the end goal of implementing the Digital-Twin Look-Up for an operationally fielded System of interacting components (Figure 12).

There 3 threads of research that must be conducted:

(1) DLNN CONFIGURATIONS: Develop a better understanding of how to configure the neural networks (i.e. hidden layers) and refine the hyper-parameters for DLNNs for the hybrid behavior
(2) SCOPE OF APPLICABLE SYSTEMS: Expand the scope to increasingly complex behavioral systems and documenting the types of systems that it works for and the systems it does not work for (and why)
(3) HOW-TO-MANUAL: Developing a How-To-Manual with examples so others can deploy the methodology

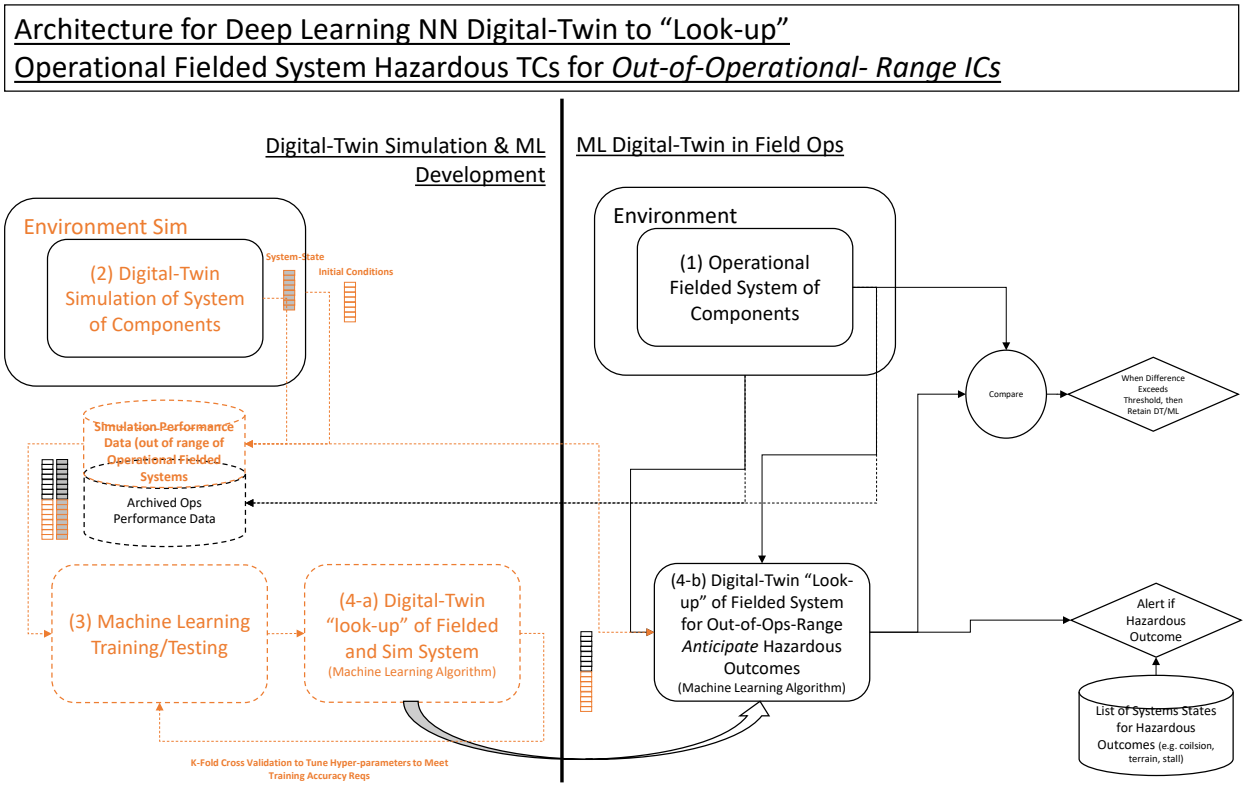


FIGURE 12: Architecture for Deep Learning Neural Network to "Look-up" System behavior. The DLNN has "learned" the underlying System behavior and can predict final states for initial conditions not in the DLNN training/testing data.

References

[1] Perrow, C. (1984) Normal Accidents: Living with High-Risk Technologies. New York: Basic Books, 1984. ISBN-13: 978-0691004129

[2] Sherry, L. and R. Mauro (2015) Functional Complexity Failures and Automation Surprises: the Mysterious Case of Controlled Flight into Stall (CFIS). In Proceedings 2014 Integrated Communications Navigation and Surveillance (ICNS)Conference, Dulles, Va. April, 2014 (2015).

[3] Sherry, L., R. Mauro (2019) Anatomy of a No-Equipment-Failed (NEF) ICNS System Malfunction: The Case of Singapore Airlines SQ-327 Runway Excursion. In Proceedings IEEE ICNS Conference 2019. Dulles, Virginia; April 9-11, 2019

[4] Breen, P.G., Christopher N Foley, Tjarda Boekholt, Simon Portegies Zwart, (2020) Newton versus the machine: solving the chaotic three-body problem using deep neural networks, Monthly Notices of the Royal Astronomical Society, Volume 494, Issue 2, May 2020, Pages 2465–2470, https://doi.org/10.1093/mnras/staa713)

[5] de Lagrange J.-L., 1772 Chapitre II: Essai sur le Probleme des Trois Corps

[6] Silver D., Huang A., Maddison C. J., Guez A., Sifre L., van den Driessche G., Schrittwieser J., Antonoglou I., Panneershelvam V., Lanctot M., et al. 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484-489.

[7] McCulloch W., Pitts W., 1943. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. 7: 115-3

[8] Rosenblatt F., 1958, The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. 65(6): 386-408

[9] Hornik K., 1991, Neural Networks, 4(2), 251-257. doi:10.1016/0893-6080(91)90009-T.

[10] Cybenko G., 1989, Mathematics of Control, Signals, and Systems, 2(4), 303-314. doi:10.1007/BF02551274.

[11] Sardana; S. S. Dashora, E. Sesha S. Kamineni; S. S. C. Tanneru, S. R. Aravelli; K. Dandamudi (2022) Deep Learning Neural Networks for Complex System Validation. Capstone Sources project supervised by Brett Berlin.