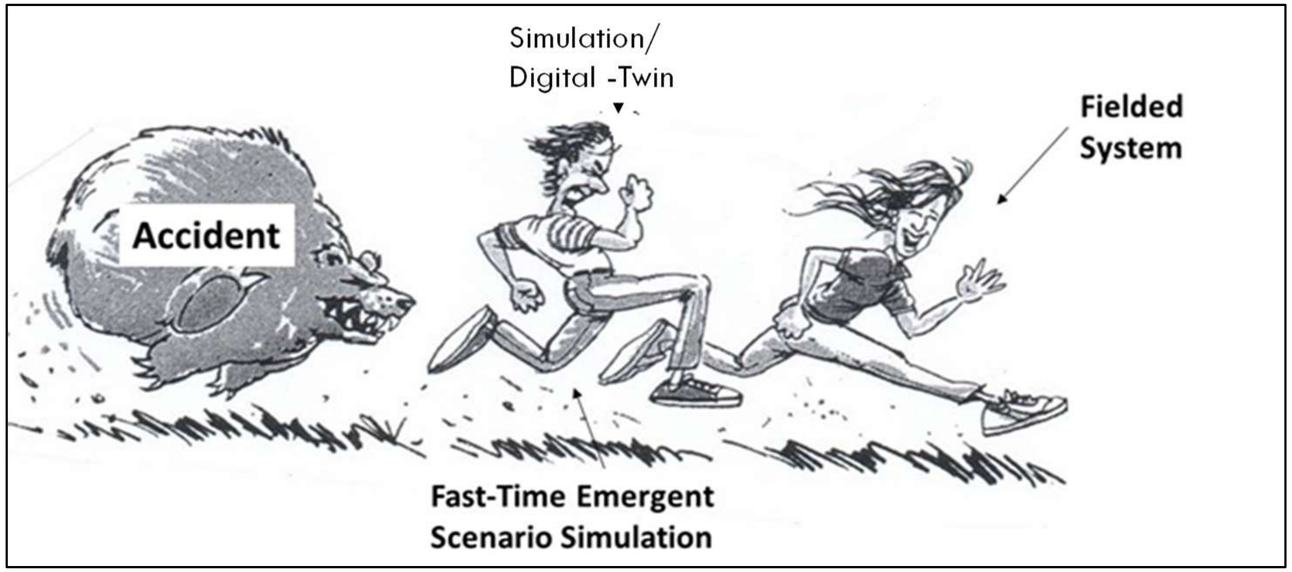
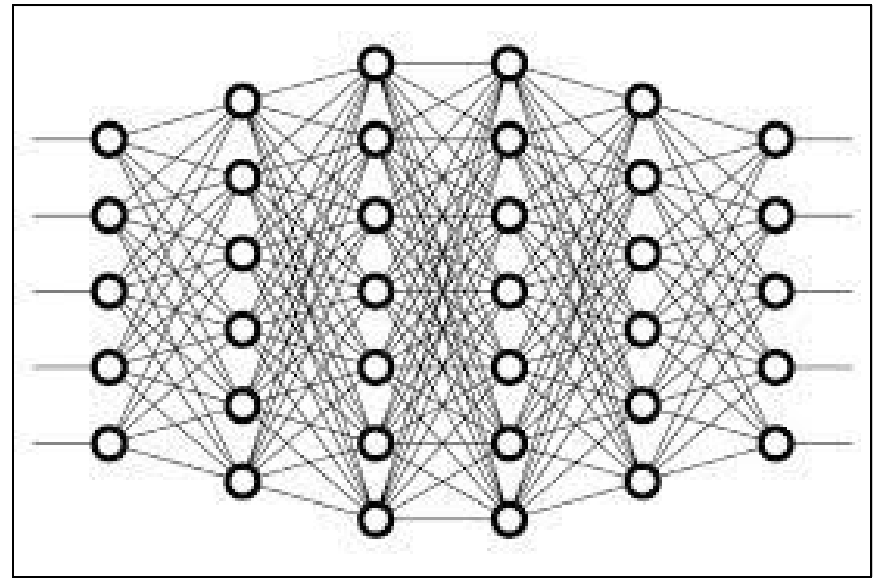




Towards the Use of Deep Learning Neural Networks for System Validation Testing of Tightly Coupled Complex Systems

Approved for public release.
Distribution is unlimited.

Lance sherry and Team
lsherry@gmu.edu

Center for Air Transportation Systems Research @ GMU

ACKNOWLEDGEMENTS:

ACQUISITION INNOVATION
RESEARCH CENTER

Holistic Assurance Framework:
Fast Time Emergent Scenario Simulation (FTESS):
potential for using deep learning neural networks for system validation testing

(WRT-1049.8.6)

Sponsors: DOT&E (R. O'Toole, S. Hobson); A&S/AE (D. Cadman)

RESEARCH TEAM – BIO(S)

Ms. Jomanah Bashatah
Ph.D. Candidate, System Engineering

- MBSE, Digital-Twin, Simulation

Ms. Amy Rose
Ph.D. Candidate System Engineering

- Digital Engineering, Remote sensing validation

Mr. Wyatt Mingus
B.Sc. System Engineering

- Machine Learning, Digital Twin

Dr. Ali Raz
Assistant Professor, Systems Engineering and Operations Research, College of Engineering and Computing

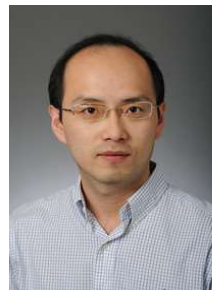
- MBSE, Digital-Twin

Dr. James Baldo
Associate Professor, Director of Master of Science in Data Analytics Engineering Program

- Software Development, Software Testing, Dev Sec Ops

Brett Berlin
Instructor Master of Science in Data Analytics Engineering Program

- Super-Computing, Data Analytics



Dr. Jie Xu
Associate Professor with the Department of Systems Engineering & Operations Research (SEOR)

- artificial intelligence (AI)/machine learning, digital twin/computer simulation

Dr. Ran Ji
Assistant Professor with the Department of Systems Engineering & Operations Research (SEOR)

- Distributionally Robust Optimization

Stochastic Programming

Dr. John Shortle
Professor and Chair, Systems Engineering and Operations Research

- Safety Risk Assessment, Rare-event Simulation, Queueing Theory

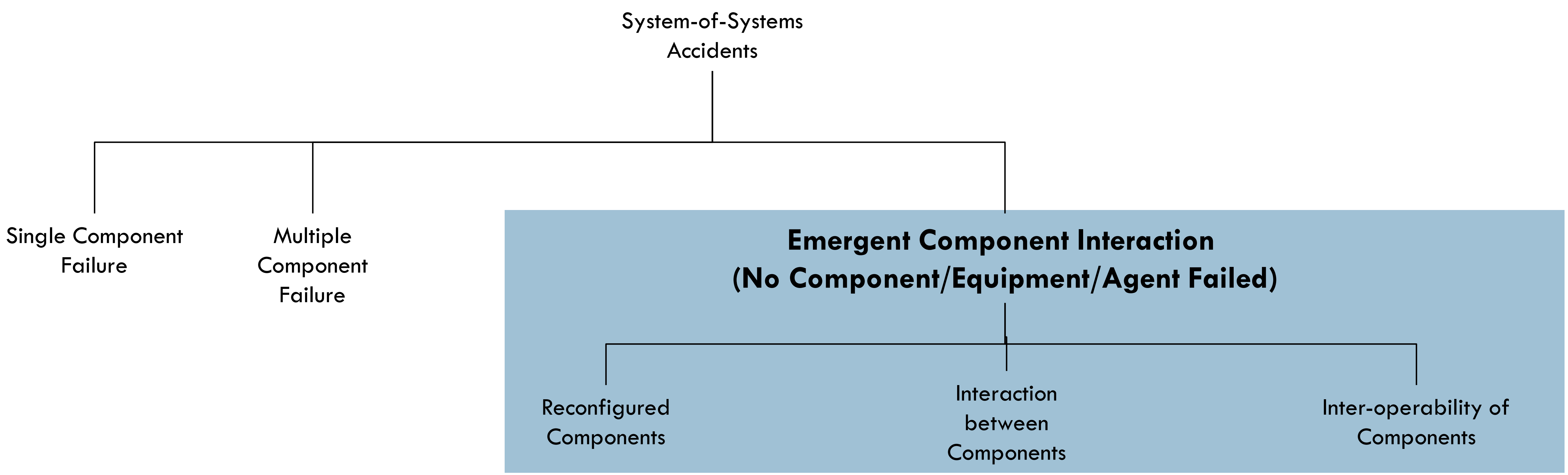
PROJECT SUMMARY

- Project Overview
  1. Analysis of modern accidents/incidents showed that no component(s) failed!
  2. Instead, modern accidents/incidents are increasingly the results of the *emergent behavior* resulting from the *interaction* of increasingly complex components of systems that are *tightly-coupled*
  3. The *combinatorics of system component* interactions *over time* makes complete testing of full coverage of the operational state-space, using agent-based simulation/digital-twin models, time and cost prohibitive
  4. Project evaluated the feasibility of using Deep Learning Neural Networks (DLNN) to generate scenarios beyond those generated by agent-based simulation/digital-twin models (i.e. supplement simulation results)
- Key Finding
  1. Deep Learning Neural Networks (DLNN) *can successfully* be used to generate scenarios for System Validation Testing beyond the range of scenarios generated by agent-based simulation models (for the class of system tested)
  2. Success achieved for Hybrid (i.e. logical and continuous behavior) systems with finite and/or repeatable behavior
  3. DLNN can be used as a "look-up" table for Digital-Twin (i.e. emergent behavior resulting from initial conditions)

Background – Accident Categories

Not all accidents/mishaps caused by **component failures**

- Anatomy of "No-Equipment Failed" Malfunctions (Sherry, Mauro, 2014, 2017a; 2017b, 2018, 2019)



Background – Component Interaction Accidents

"Normal Accidents" Perrow (1984)

- Functional Interaction Complexity Failures/Malfunctions (FICFs) (Sherry et. al., 2014 -20)

All components work as designed

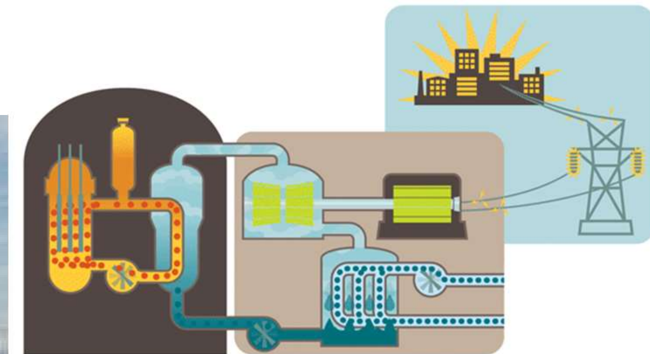
- No component *FAILED*
- Component or system migrated into hazardous operating region

"Normal Accident" Criteria:

1. The System behavior is complex (moded logic *and* continuous)
2. The System is composed of tightly coupled components
3. *Interactions occur over time*
4. The System has catastrophic potential when operating in a hazardous operating regime

"Normal Accident" Scenario

1. Start the fire
2. Disable the fire extinguisher
3. Provide ambiguous cues (that prevent intervention)

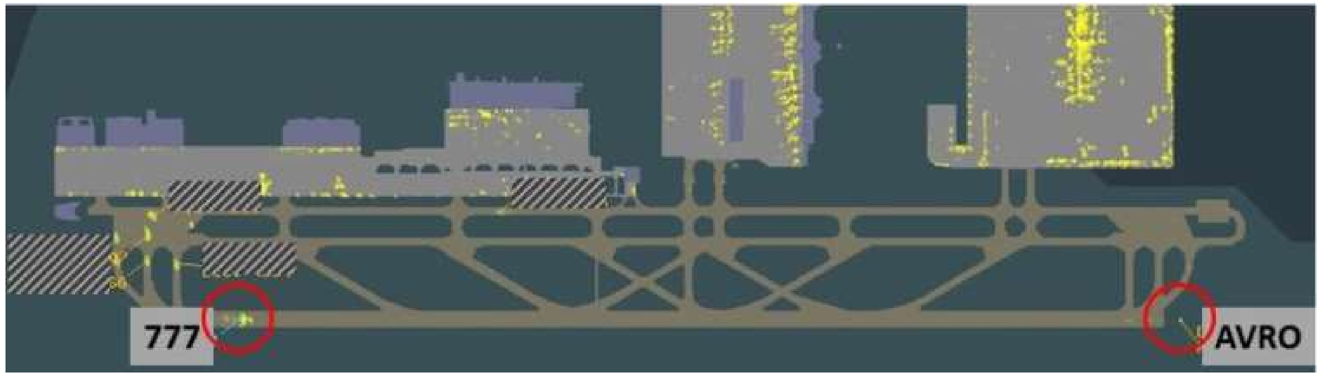


Three Mile Island Nuclear Reactor

Munich Airport Runway Excursion

BACKGROUND: MUNICH AIRPORT RUNWAY EXCURSION

1. To accommodate A380, airport moves Localizer antennae away from runway end (changes ILS Critical Area)
2. Low Visibility conditions causes long departure queue
3. Air Traffic Controller, trying to expedite departures, clears Avro for mid-runway takeoff
4. Air Traffic Controller clears SQ237 for approach
5. 777 decides to “practice” CAT III automatic landing
6. Avro takeoff roll to end of runway and lift-off
7. Localizer signal is deflected (due to Avro)
8. 777 Automatic Landing System follows deflected Localizer signal and lands adjacent the runway
9. 777 weight-on-wheels inhibits Go Around button selection by flight-crew to intervene



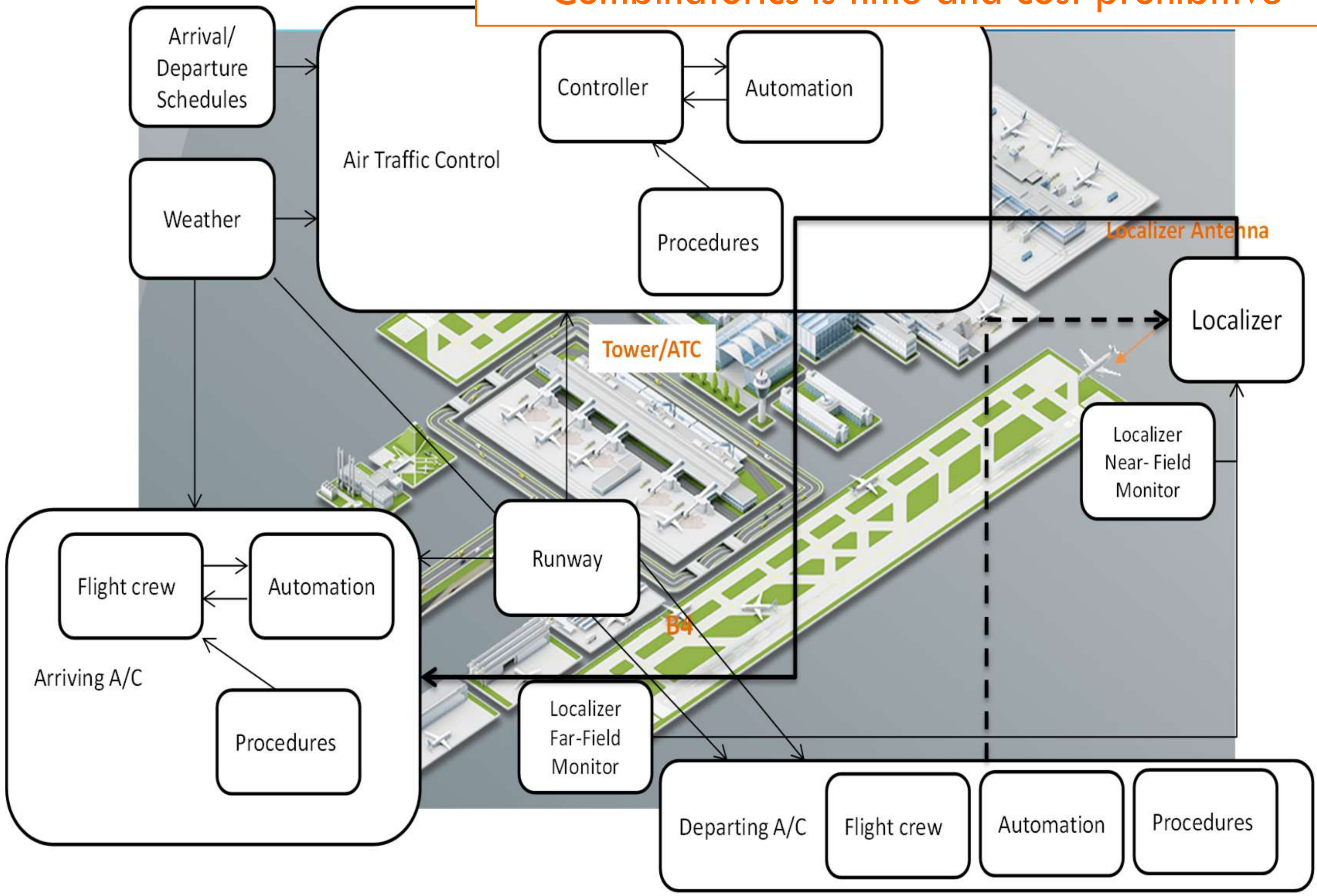
Failure of (designers) imagination to prevent?

Background: Munich Airport Runway Excursion Simulation Model

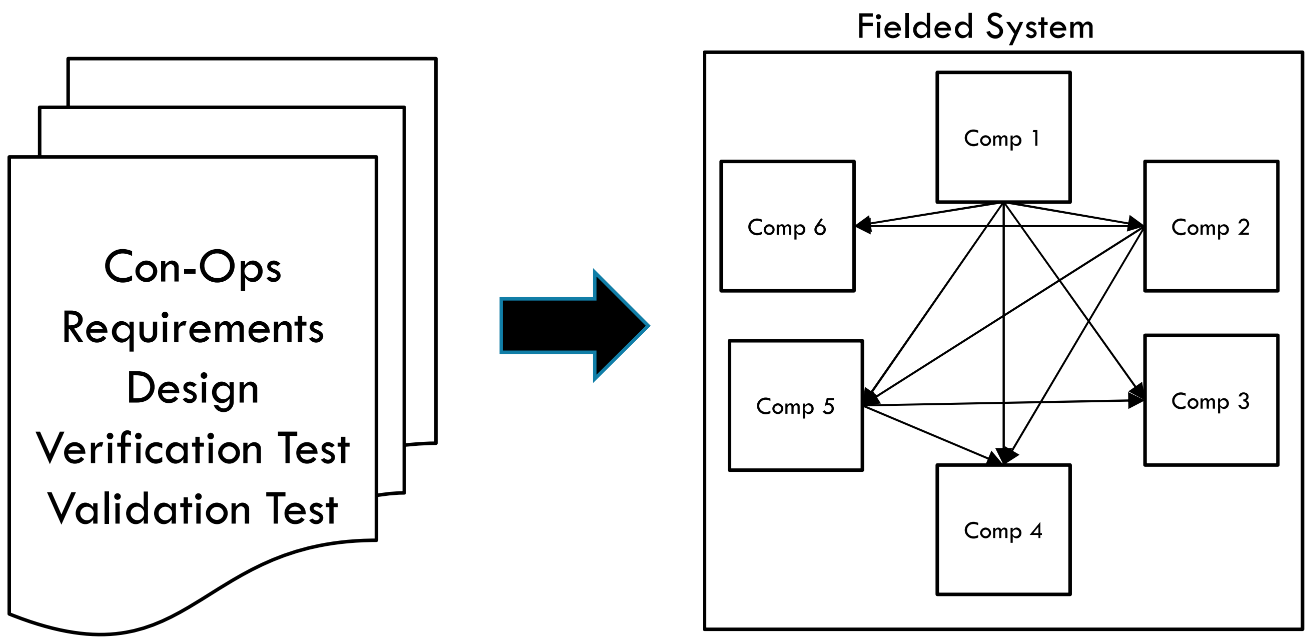
System Components:

1. Air Traffic Control
    1. Procedures
    2. Automation
    3. Controller
2. Departing Aircraft
    1. Procedures
    2. Automation
    3. Flight crew
3. Arriving Aircraft
    1. Procedures
    2. Automation
    3. Flight crew
4. Airport Arrival/Departure Schedule
5. Weather
6. Runway
7. Localizer
    1. Localizer Near-field Monitor
    2. Localizer Far-field Monitor

- 15 components
- Running sim for *all plausible* Initial Condition Combinatorics is time and cost prohibitive

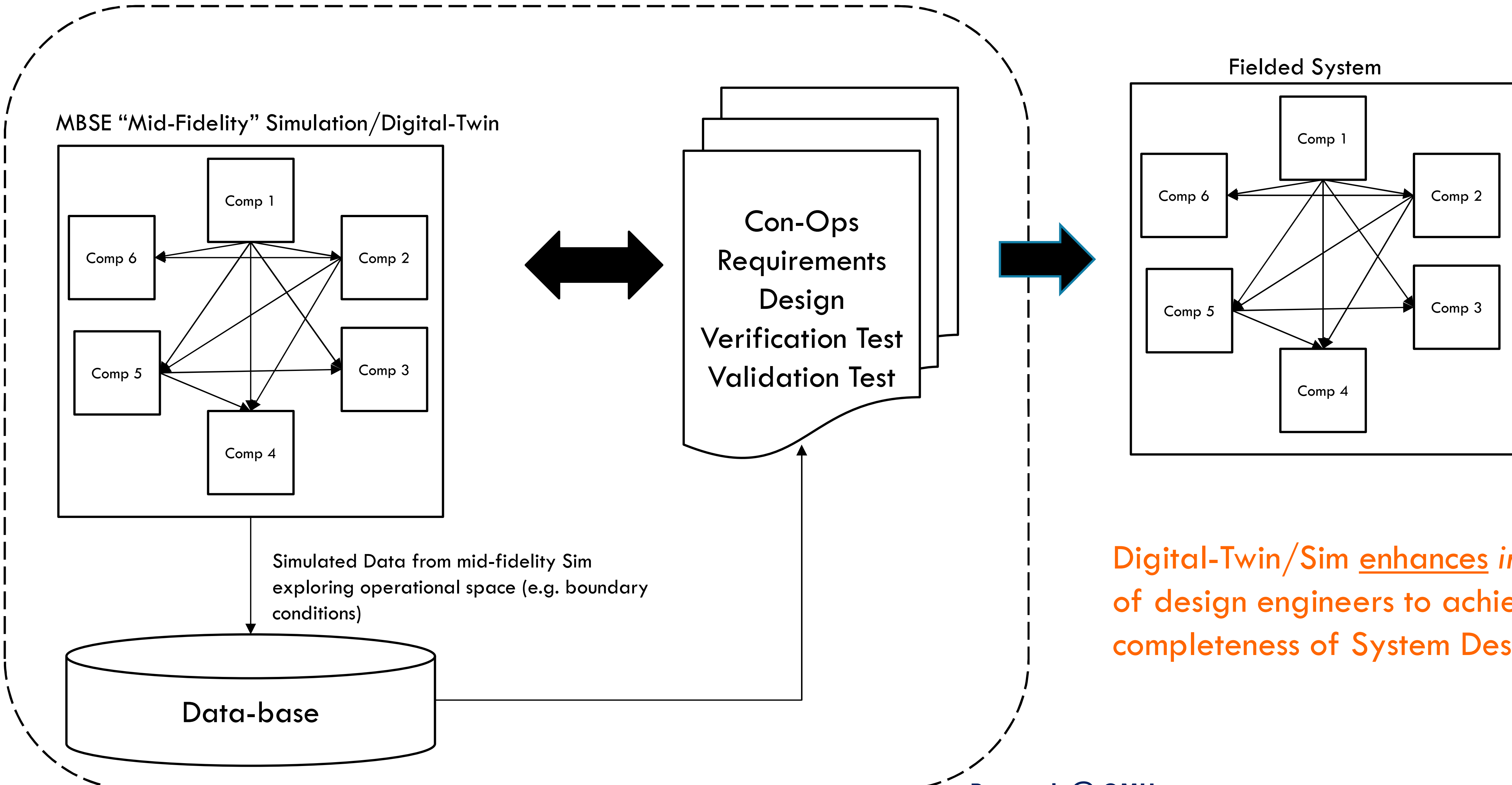


Traditional System Development



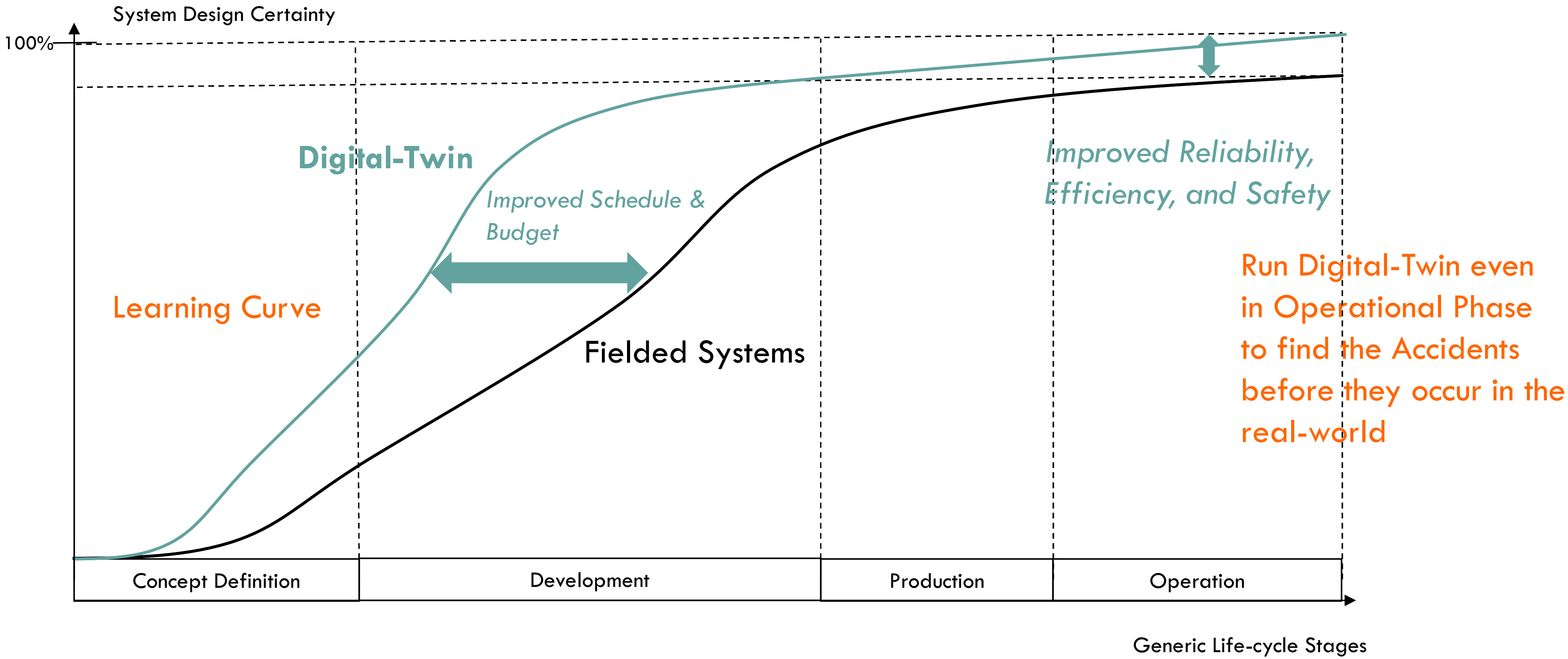
Completeness of System Design is dependent on *imagination* of design engineers

TRADITIONAL MODEL-BASED/DIGITAL-TWIN SYSTEM DEVELOPMENT



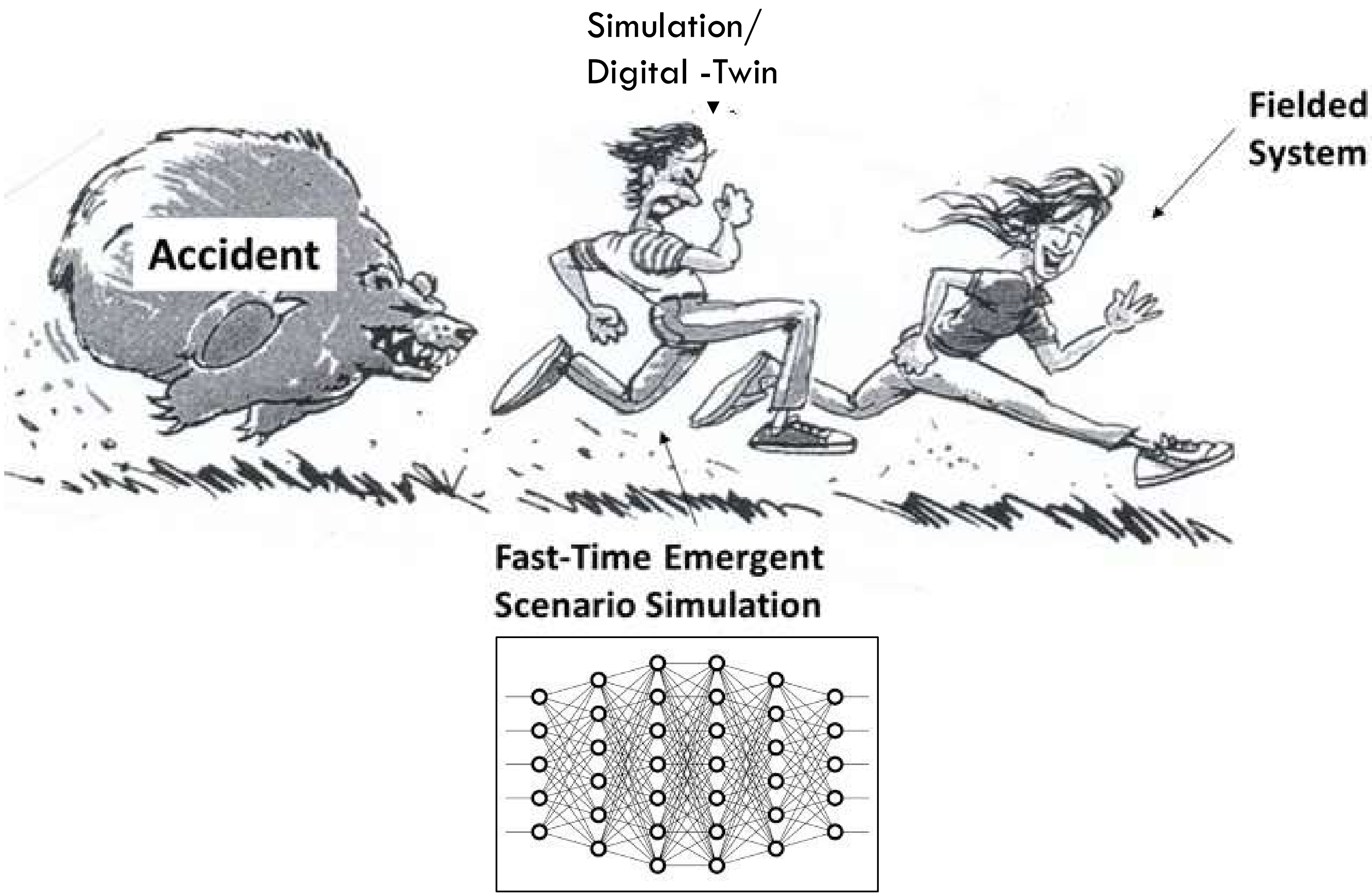
Digital-Twin/Sim enhances *imagination* of design engineers to achieve completeness of System Design

DEVELOPMENT LIFE-CYCLE



Berlin (2021) Personal Communications

CON-OPS – POKE THE ACCIDENT BEAR EARLY AND OFTEN



Limitations of Simulation

- Combinatoric Explosion
  - Component Initial States
  - Time-dependence
- Running Simulation to end state is time and cost prohibitive

- **Can we use DLNN to speed-up/reduce cost of System Validation Testing**?
  - Continuously uncovering emergent rare-events without simulation cost/time
  - Increase operational Initial State Coverage

Con-Ops: Using DLNNs for System Validation Testing

Step 1: Collect System X behavior data for all $x_{i,j}(t)$ for as many scenarios as possible

- Data from operations and/or simulation
- Note: By definition this data set is *subset* of *all* possible Initial/Terminal Condition pairs

Step 2: Develop (i.e. train/test) DLNN using available Initial/Terminal Condition pairs data set

Step 3: Calculate "confidence interval" for DLNN to correctly predict Initial/Terminal Condition pairs not in development data set

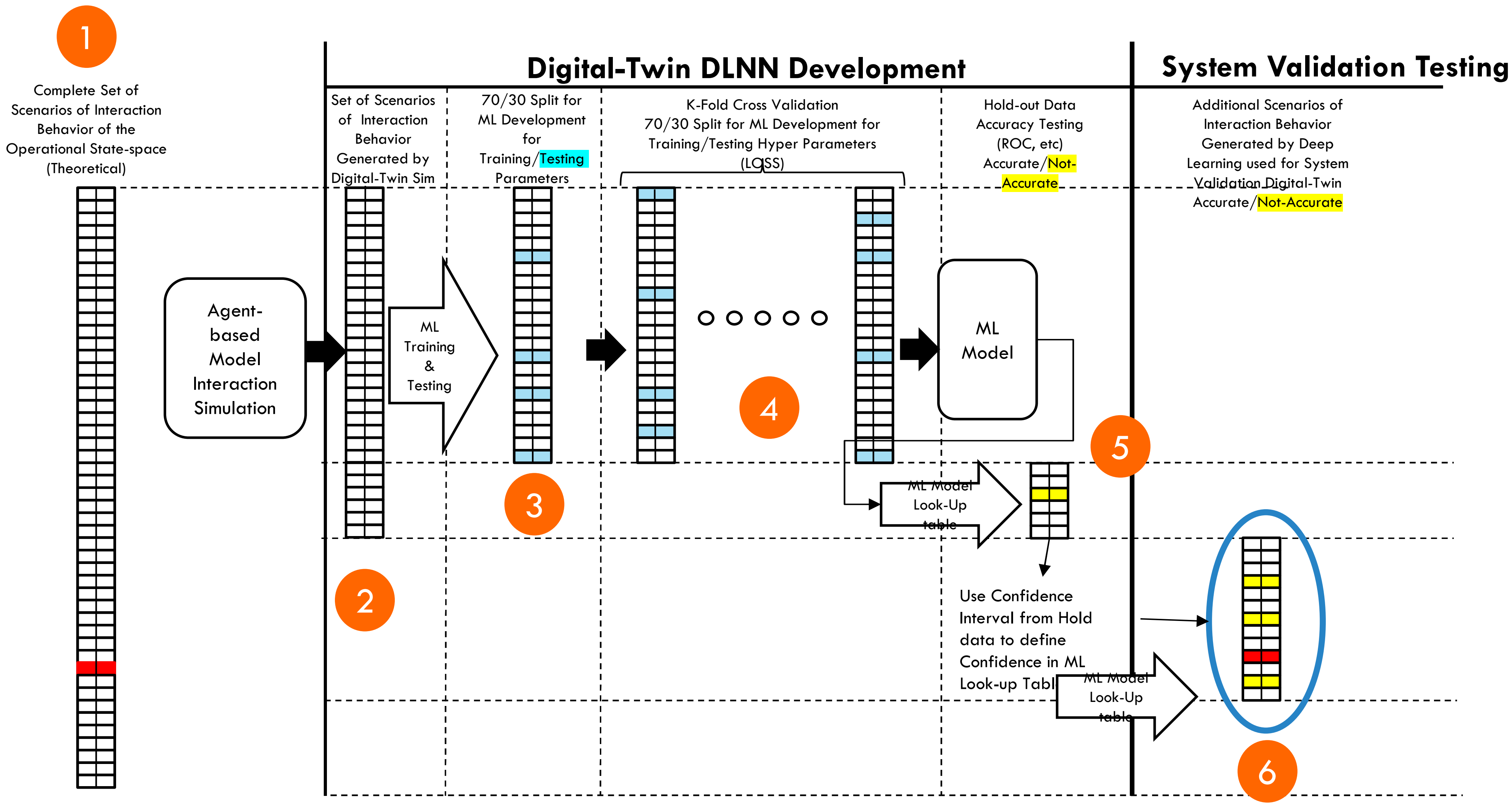
Step 4: Use DLNN as a "look-up table" to test *all* possible Initial/Terminal Condition pairs

- Keep testing and get as close to *all* combinations as possible

Step 5: For Initial/Terminal Condition pairs that are deemed "unsafe" by DLNN check on simulation/analysis

Step 6: Continuously check accuracy of DLNN using live operational data and re-train when no longer "calibrated"

Con-Ops: Using DLNNs for System Validation Testing



Combinatorics of component interaction and timing prohibit generating complete operational-space

Additional interaction scenarios to add to coverage of the operational-space (that cannot be generated by sim). Note: some results may be inaccurate

CASE STUDY: SYSTEM OF COUPLED COMPONENTS

Behavior of Component is hybrid:

- Moded (dependent on logic)
- Continuous dynamics
  - i.e. depending on the mode, a different continuous behavior occurs

Vessel filled with a gas part of Refinery process

Component States

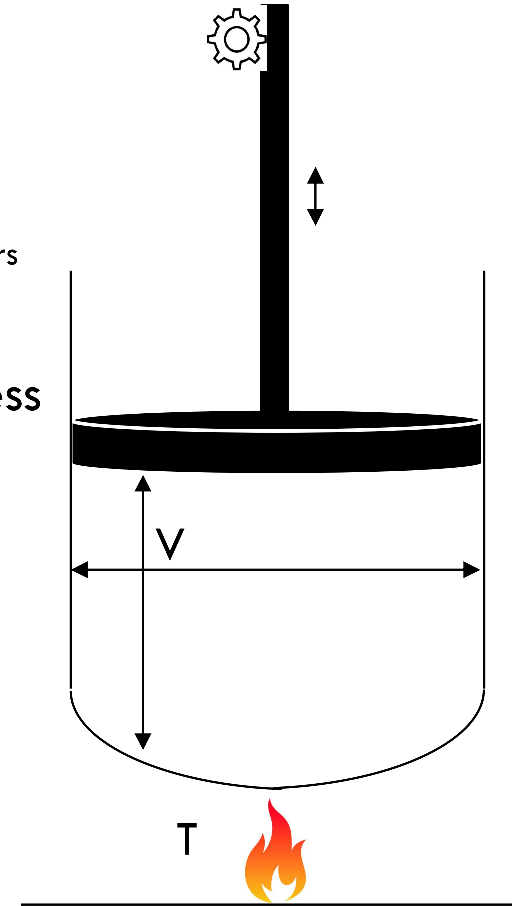
- Temperature
- Volume

Emergent State:

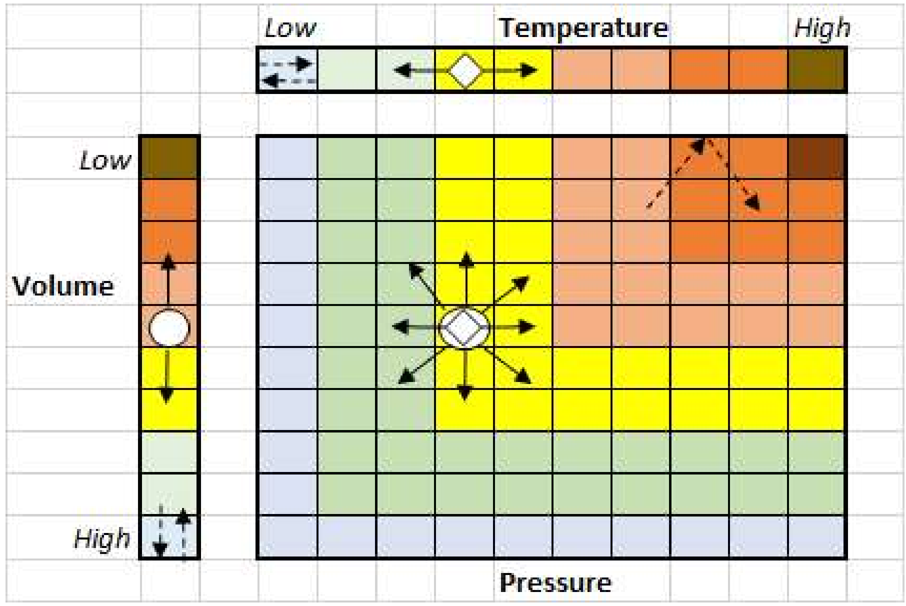
- Pressure

Hazard:

- Pressure in excess of vessel material strength

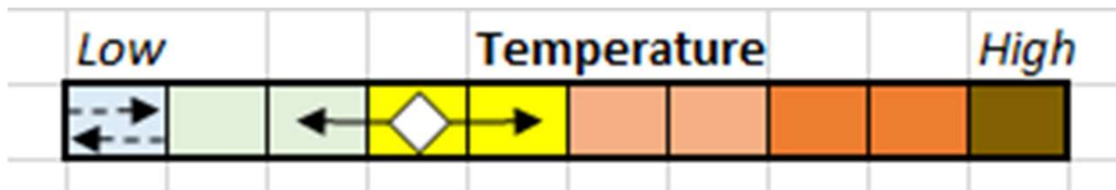


$$P = nRT/V$$

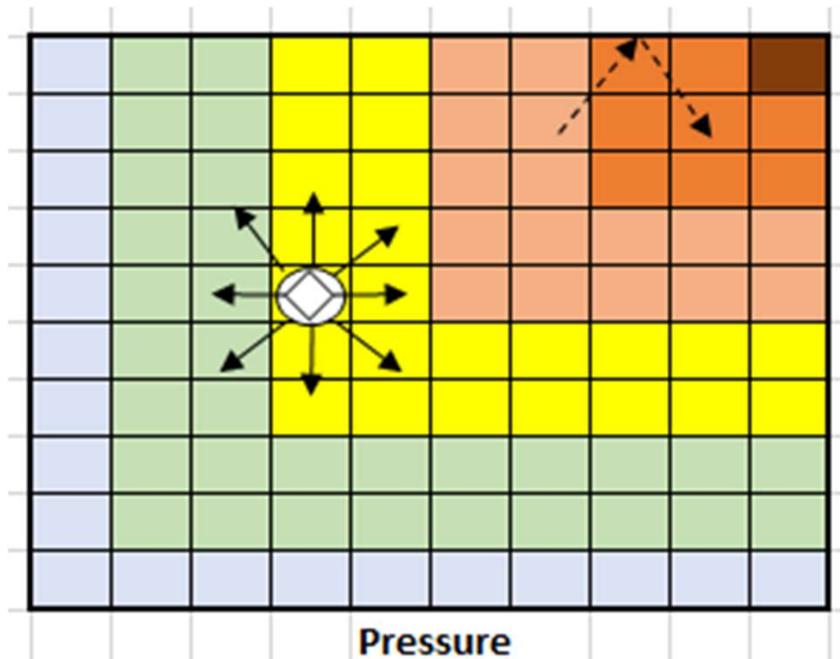


CASE STUDY: DLNN FOR SYSTEM VALIDATION TESTING

1 x 10



10 x 10

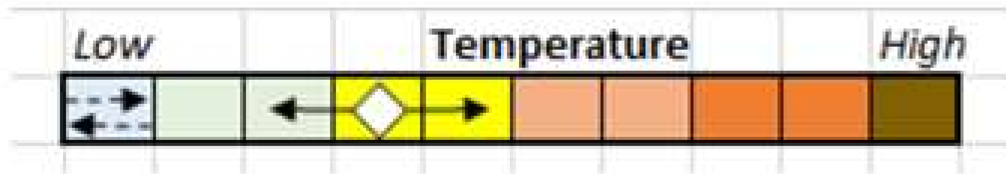


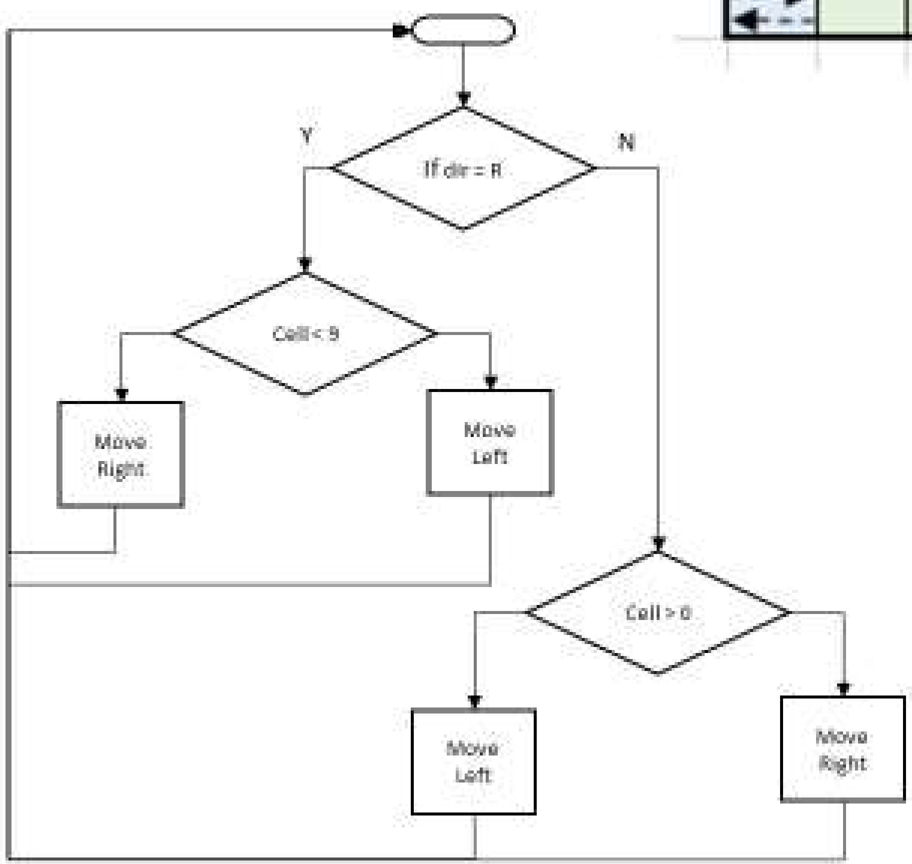
Note:
Behavior is hybrid
Discrete
Logic/Continuous

"Bounce" represents
Discrete Logic
Movement represents
continuous

Case Study: 1 x 10

- 1 x 10





initial	direction	final
0	R	7
1	R	6
2	R	5
3	R	4
4	R	3
5	R	2
6	R	1
7	R	0
8	R	1
9	R	2
0	L	7
1	L	8
2	L	9
3	L	8
4	L	7
5	L	6
6	L	5
7	L	4
8	L	3
9	L	2

Note:
Behavior is hybrid
Discrete
Logic/Continuous

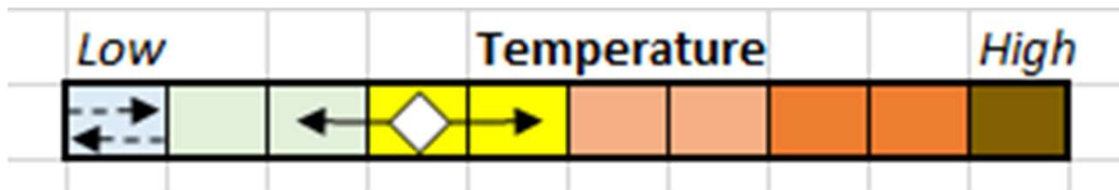
"Bounce" represents
Discrete Logic
Movement represents
continuous

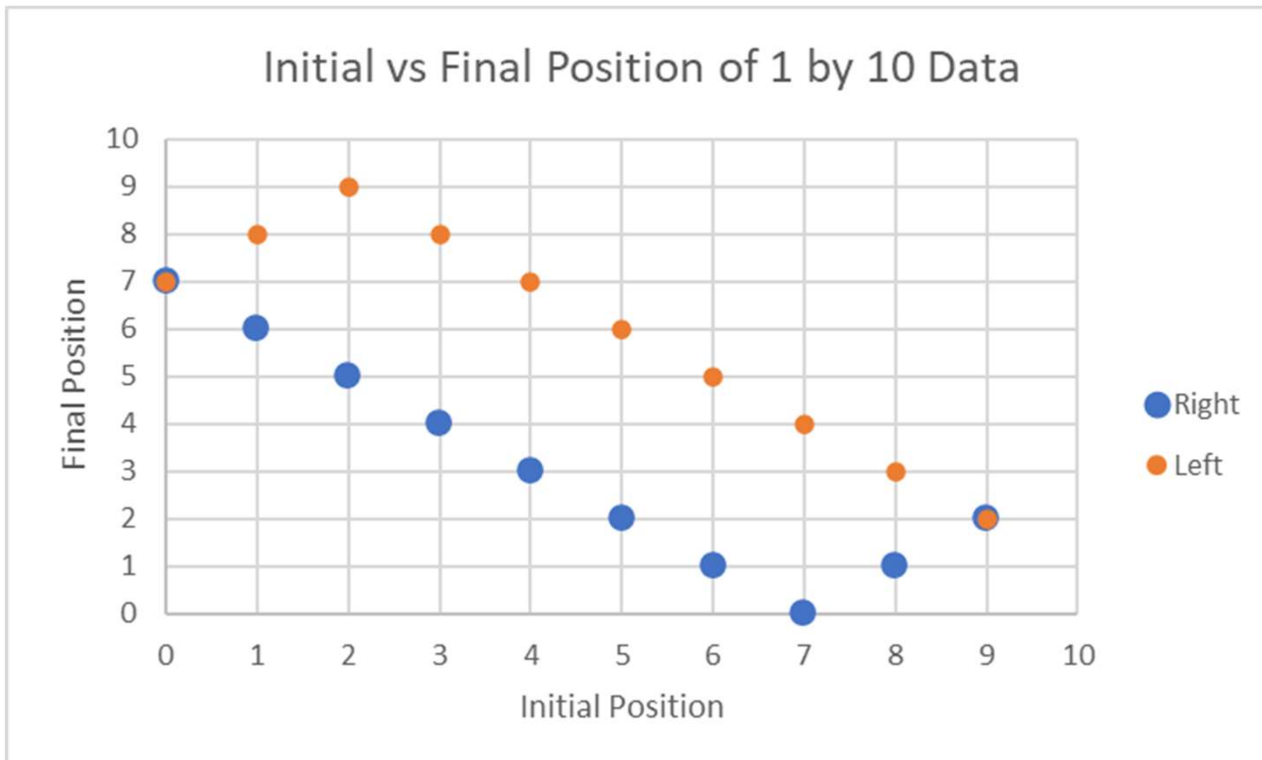
Case Study: 1 x 10

1 x 10

Training Data

initial	direction	final
0	R	7
1	R	6
2	R	5
3	R	4
4	R	3
5	R	2
6	R	1
7	R	0
8	R	1
9	R	2
0	L	7
1	L	8
2	L	9
3	L	8
4	L	7
5	L	6
6	L	5
7	L	4
8	L	3
9	L	2





Note:
Behavior is hybrid
Discrete
Logic/Continuous

"Bounce" represents
Discrete Logic
Movement represents
continuous

CASE STUDY: SYSTEM OF COUPLED COMPONENTS

```
import pandas as pd
import tensorflow as tf
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
sc = StandardScaler()
import statistics
```

DLNN created with TensorFlow and Keras libraries

Scikit-learn, pandas, and numpy used for data processing

Statistics package used for data analysis

CASE STUDY: SYSTEM OF COUPLED COMPONENTS

Initial and direction columns used as input values (X)

Final position column used as output values (Y)

Train/Test follows 70/30 split

He_uniform initializer

Rectified Linear Unit (relu) activation

Scikit-learn StandardScaler.transform function

```
#####assigning train and test #####
X_train, X_test_un, Y_train, Y_test = train_test_split(X, Y, random_state=42, test_size=tsiz) #, stratify=Y)
X_test_dup = X_test_un
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test_un)
X_full_test = sc.transform(X_full_test_un)
#print(Y_test.shape[0],Y_train.shape[0])


###BUILD THE MODEL###
ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #1
ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #2
ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #3
#ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #4
#ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #5
#ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #6
#ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #7
#ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #8
#ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #9
#ann.add(tf.keras.layers.Dense(den, input_dim= 2, kernel_initializer='he_uniform', activation='relu')) #10
ann.add(tf.keras.layers.Dense(1))
ann.compile(optimizer="adam", loss='mae', metrics=['accuracy'])
```

```
###TRAIN AND RUN THE MODEL###
ann.fit(X_train, Y_train, epochs=epo, verbose=0)
test_loss, test_accuracy = ann.evaluate(X_test, Y_test)
print("test loss, test accuracy",test_loss,test_accuracy)
Y_pred = ann.predict(X_test)
Y_full_pred = ann.predict(X_full_test)
```

CASE STUDY: SYSTEM OF COUPLED COMPONENTS

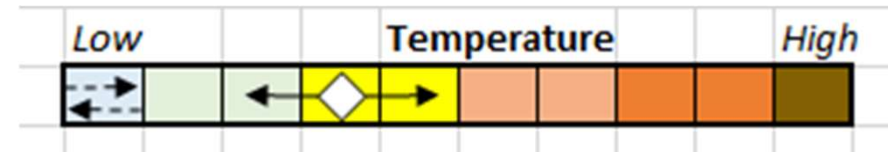
Values of predicted and expected output compared to determine true accuracy

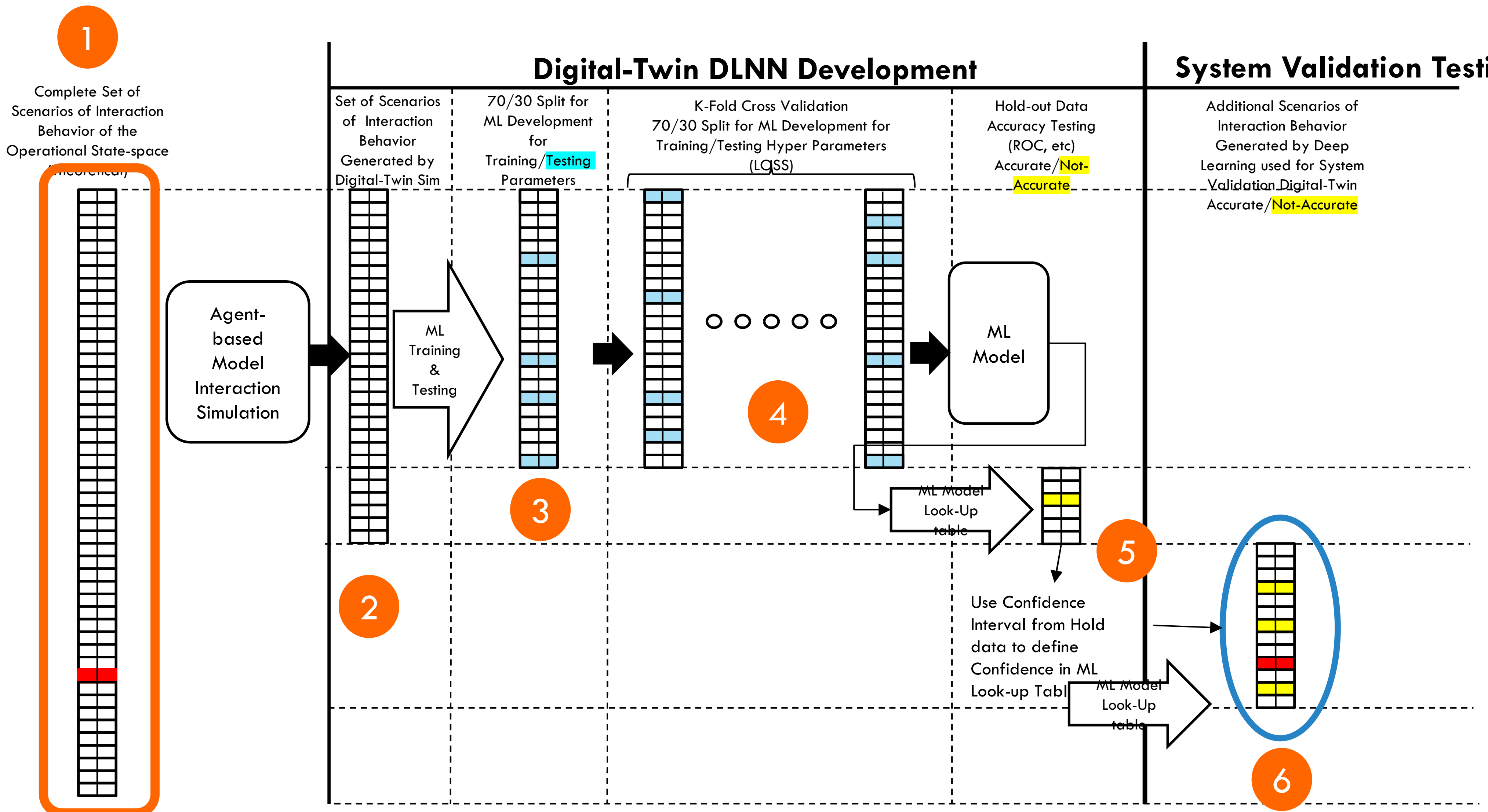
Accuracy values then stored for all runs in a .csv file to be analyzed later

```
Y_test = pd.DataFrame(Y_test)
Y_full_test = pd.DataFrame(Y_full_test)
pred_test_df = pd.concat([Y_pred, Y_test],axis=1)
pred_test_df.columns=['Y_pred','Y_test']

###Calculationg Accuracy###
pred_test_df['correct_prediction'] = np.where(pred_test_df.iloc[:,0] == pred_test_df.iloc[:,1], 1, 0)
pred_full_df['correct_prediction'] = np.where(pred_full_df.iloc[:,0] == pred_full_df.iloc[:,1], 1, 0)
accuracy_hiddenset = pred_test_df['correct_prediction'].sum()/len(pred_test_df['correct_prediction'])
accuracy_fullset = pred_full_df['correct_prediction'].sum()/len(pred_full_df['correct_prediction'])
print('test Accuracy: %f',accuracy_hiddenset, accuracy_fullset) #, acc_val)
```

DEVELOP DLNN FOR 1 X 10



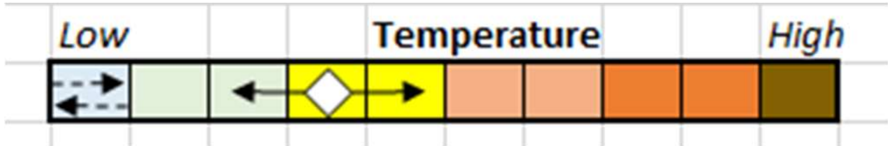


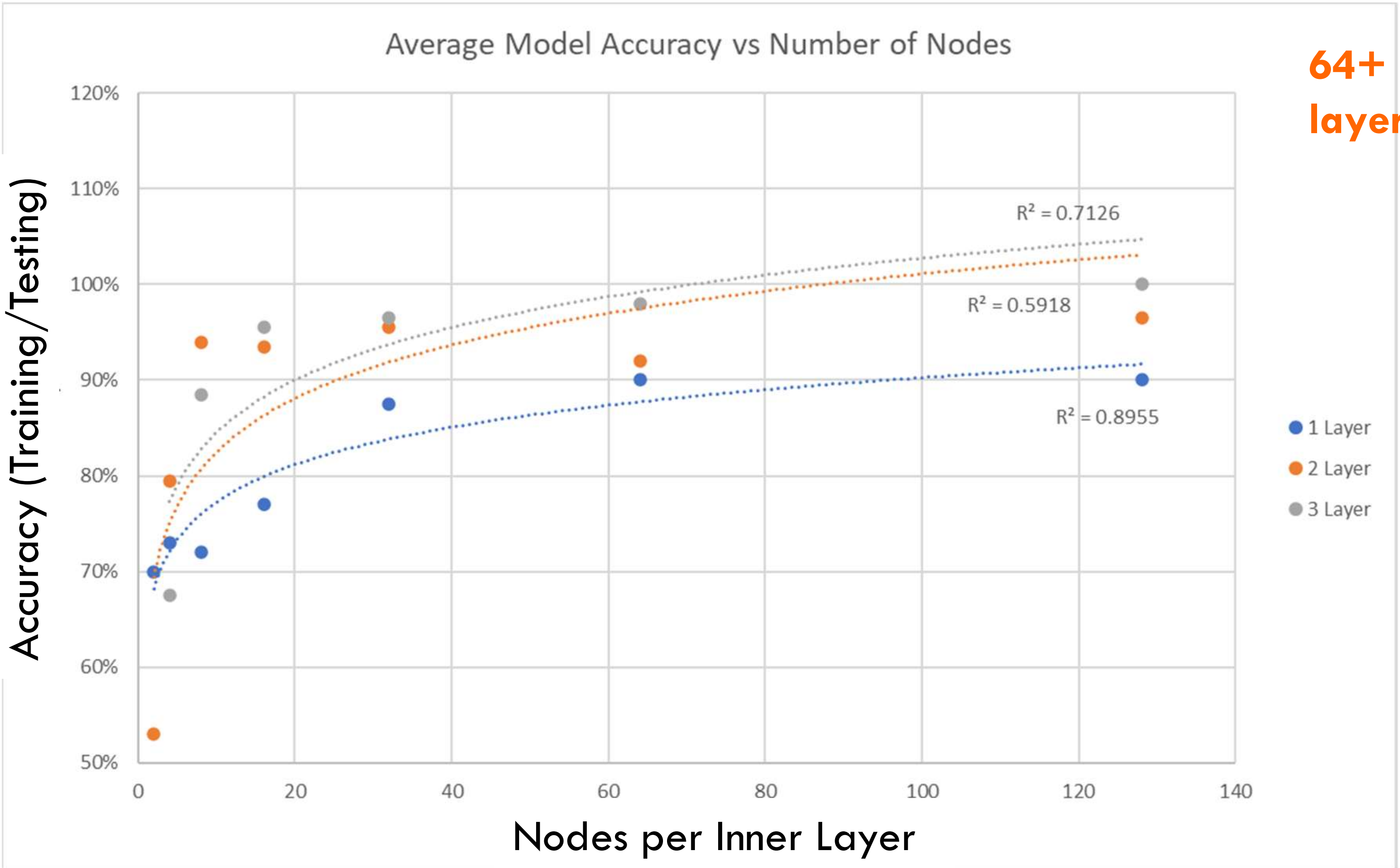
Combinatorics of component interaction and timing prohibit generating complete operational-space

Additional interaction scenarios to add to coverage of the operational-space (that cannot be generated by sim). Note: some results may be inaccurate

Develop DLNN for Full Data Set

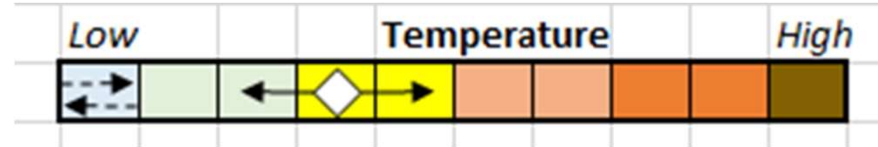
1 X 10: DLNN Configuration Performance

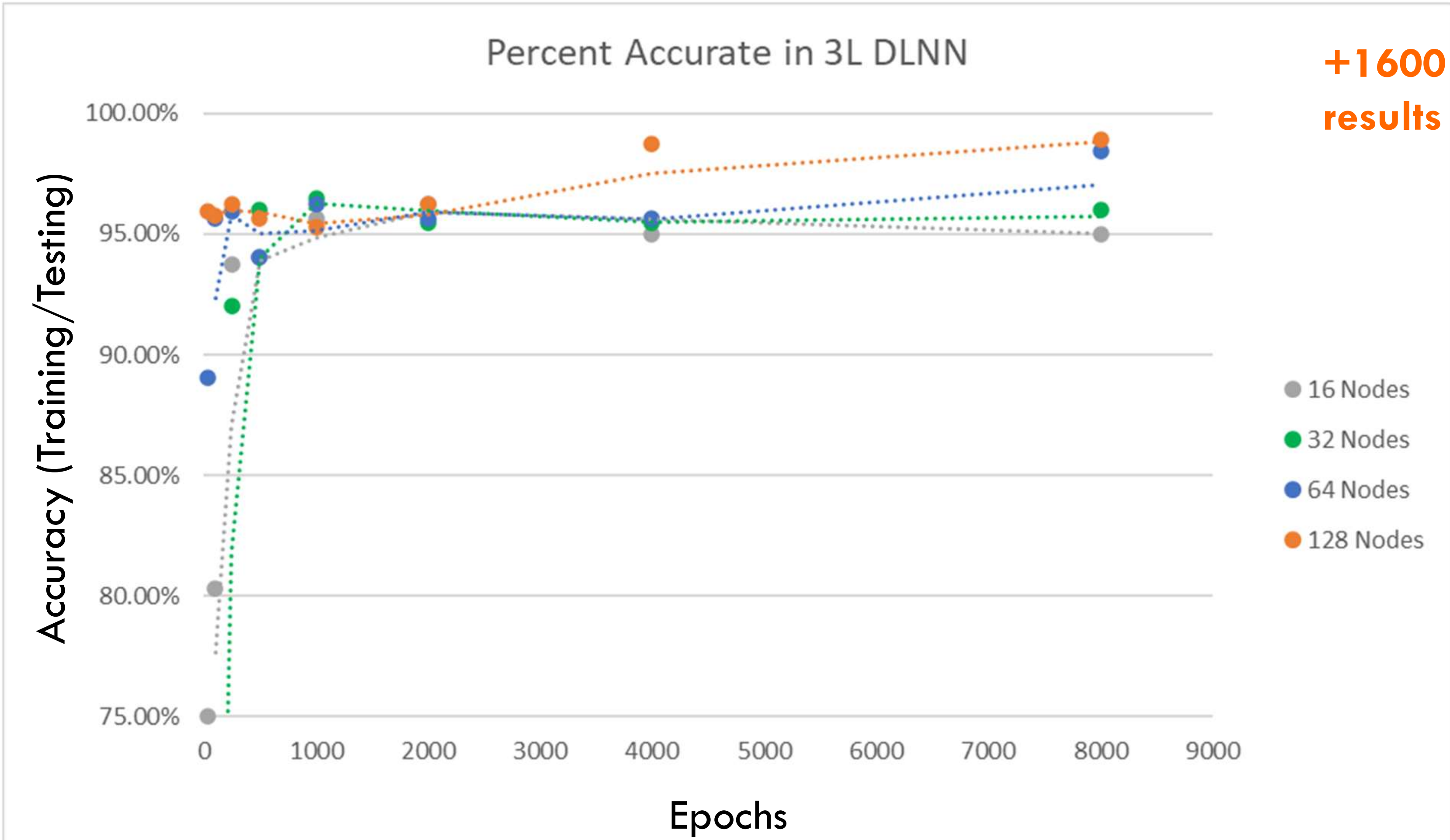




64+ Nodes per Inner layer best results

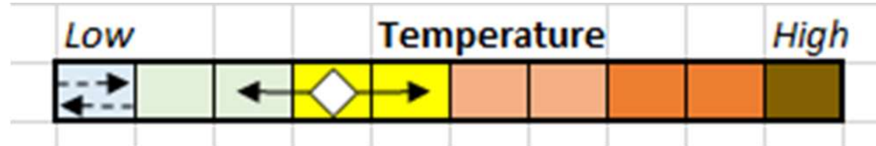
1 X 10: DLNN Configuration Performance





+1600 Epochs best results

BASELINE: FULL DATA SET



Optimal model: 64 Nodes, 3 Layers, 16000 Epochs

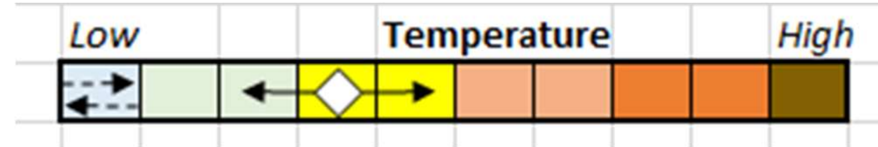
- Marginal gains with128N, 3L, 4000E, more consistent

Build 100 DLNNs (3L6N16000E) with Full Data Set (i.e. 20 target/feature pairs, no duplicates)

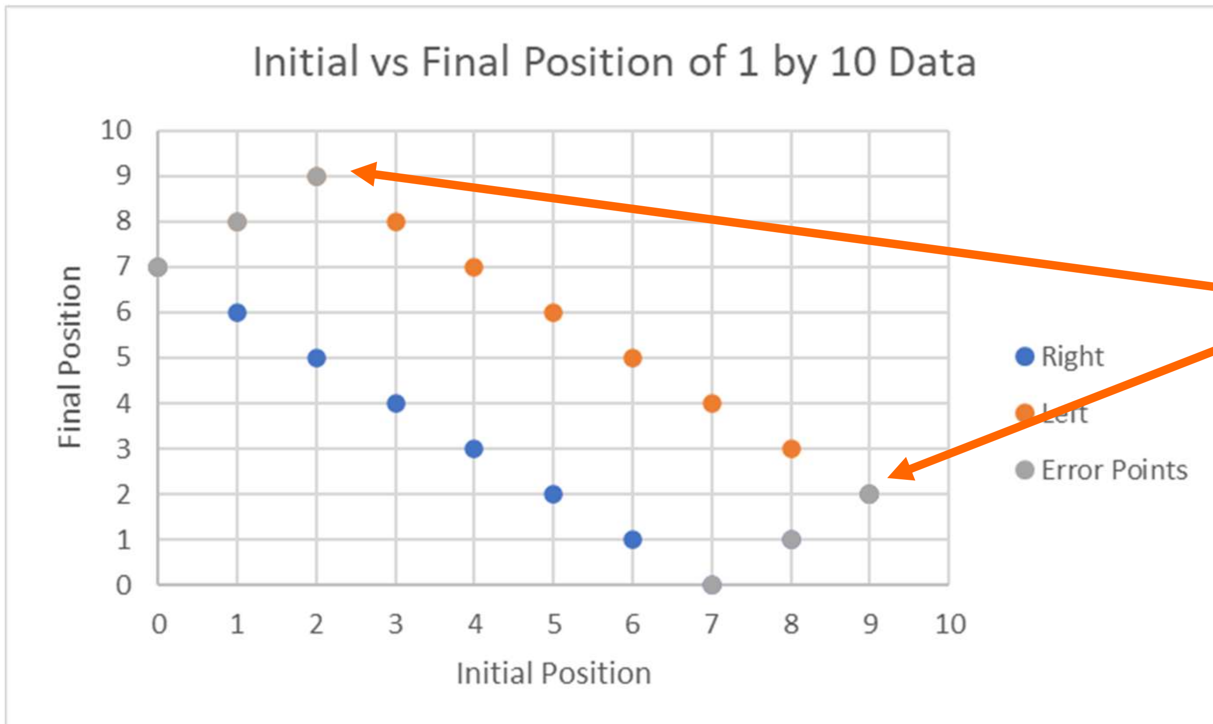
71 out of the 100 achieved a 100% Training/Testing Accuracy

Scenario		DLNN Training/Testing Accuracy					
3L64N16000E	Data for Training	Average	Median	Min	Max	Std. Dev	Count 100%
Full	20	98%	100%	90%	100%	2.53%	71

1 X 10 EXPERIMENT



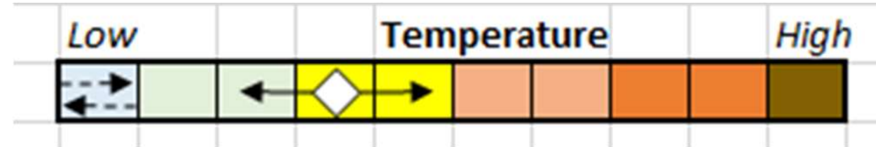
Optimal model: 64 Nodes, 3 Layers, 16000 Epochs



"Trouble" Feature/Target Pairs

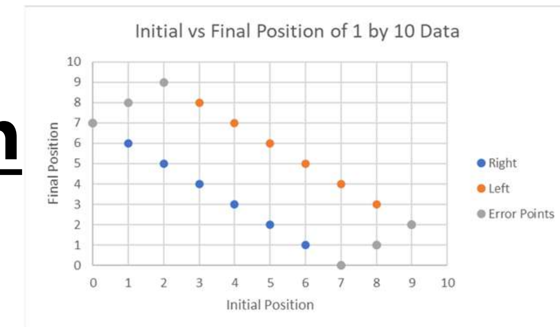
Can duplicating these data points improve accuracy?

1 X 10 Experiment



Optimal model: 64 Nodes, 3 Layers, 16000 Epochs

- Minimal difference between that and 128N, 3L, 4000E, more consistent

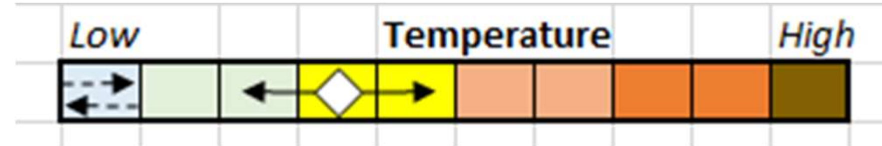


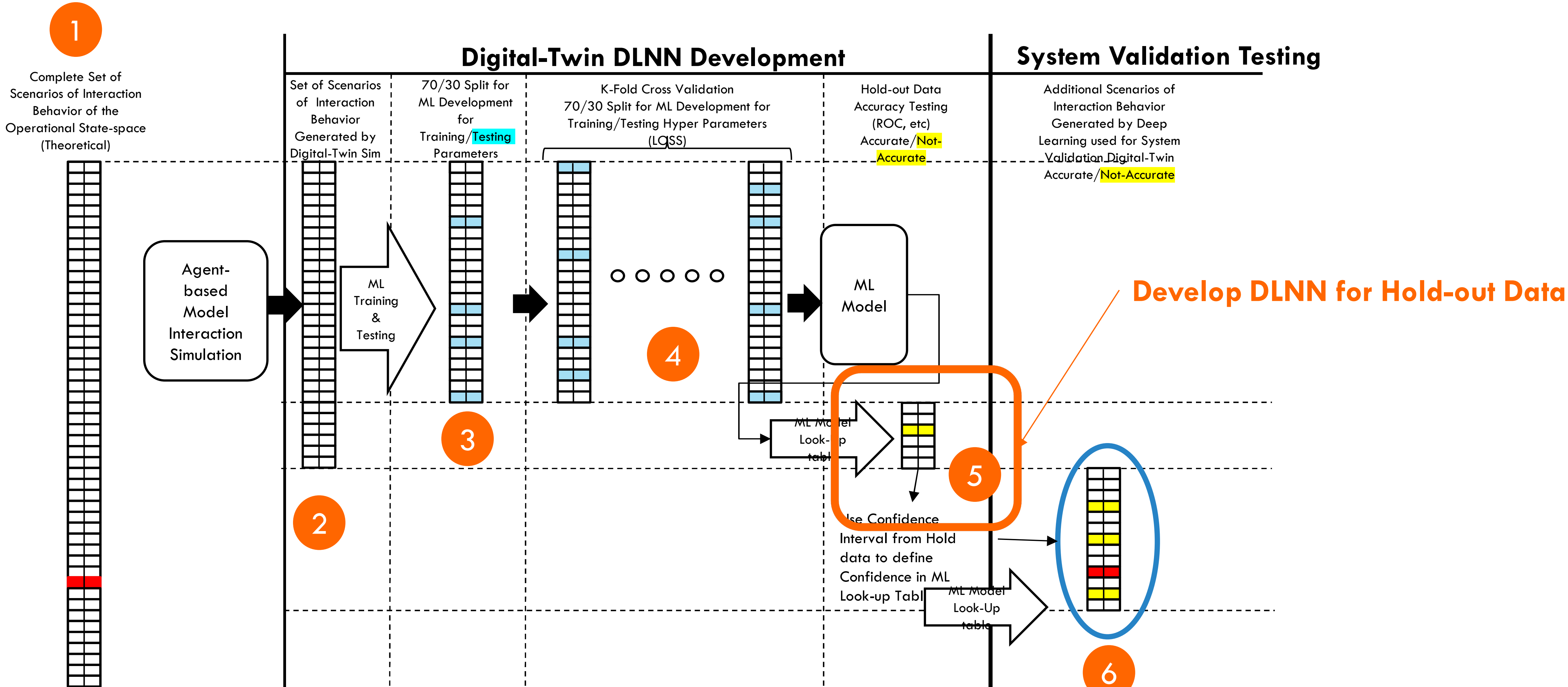
Build 100 DLNNs (3L6N16000E) with **Full Data Set with Duplicates for the "trouble" Feature/Target Pairs**

100 out of the 100 achieved a 100% Training/Testing Accuracy

Scenario		DLNN Training/Testing Accuracy					
3L64N16000E	Data for Training	Average	Median	Min	Max	Std. Dev	Count 100%
Full	20	98%	100%	90%	100%	2.53%	100

1 X 10 Experiment

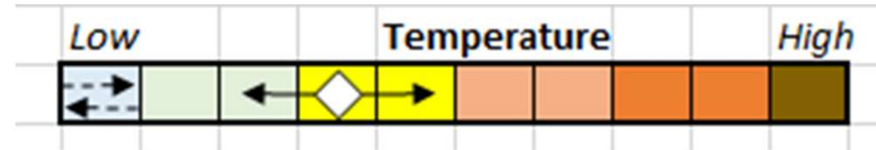




Combinatorics of component interaction and timing prohibit generating complete operational-space

Additional interaction scenarios to add to coverage of the operational-space (that cannot be generated by sim).
Note: some results may be inaccurate

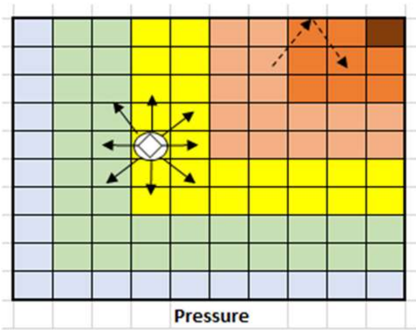
1 X 10 Experiment



1 X 10 System: 3 Inner Layers, 64 Nodes, 4000 Epochs

Experiment Scenario	Data Set	# DLNNS out of 100 with 100% Accuracy	# Feature/Targets Pairs Correctly Predicted out of 20 Using 100% Accurate DLNN
Baseline	Full Data (20)	71	20/20 (100%)
	Full Data with Duplicates for "Trouble" pairs (28)	100	20/20 (100%)
Hold-out	Full Data minus Hold Out (19)		16/20 (80%)
	Full Data with Duplicated for "Trouble Pairs" minus Hold Out (19)		19/20 (95%)

10 X 10 EXPERIMENT



10 X 10 System: 3 Inner Layers, 128 Nodes, 16000 Epochs

Experiment Scenario	Data Set	# DLNNS out of 100 with 100% Accuracy	# Feature/Targets Pairs Correctly Predicted out of 800 Using 100% Accurate DLNN
Baseline	Full Data (800)	52	800/800 (100%)
	Full Data with Duplicates for "Trouble" pairs (880)	~100	800/800 (100%)
Hold-out	Full Data minus Hold Out (800)		~640/800 (80%)
	Full Data with Duplicated for "Trouble Pairs" minus Hold Out (880)		~760/800 (95%)

Towards the Use of Deep Learning Neural Networks for System Validation Testing of Tightly Coupled Complex Systems

DLNN for System Validation

- It works!
  - At least for some tightly-coupled systems
- Expands operational Initial Conditions Coverage
  - Includes both Initial Condition Combinatorics *and* Time Dependence Combinatorics
- DLNN Operates as "Look-up Table"
  - No processing time

- Lessons Learned
  - DLNNs can "learn" underlying behavior of system
  - Not every DLNN will have 100% accuracy
    - Find one or more that do have 100% accuracy
  - Accuracy can be improved by duplicating "trouble" scenarios (with unusual behaviors)
  - Use "ensemble approach" by using multiple DLNNs

Future Work

- What classes of systems will it work for?
  - Scale for complexity
- How to calculate the "Confidence Region" for the Hold Out Data?
  - Wasserstein Distance?
- User Manual so (even) System Engineers can develop DLNN

lsherry@gmu.edu