



AI with Model-Based Design: *Virtual Sensor Modeling*

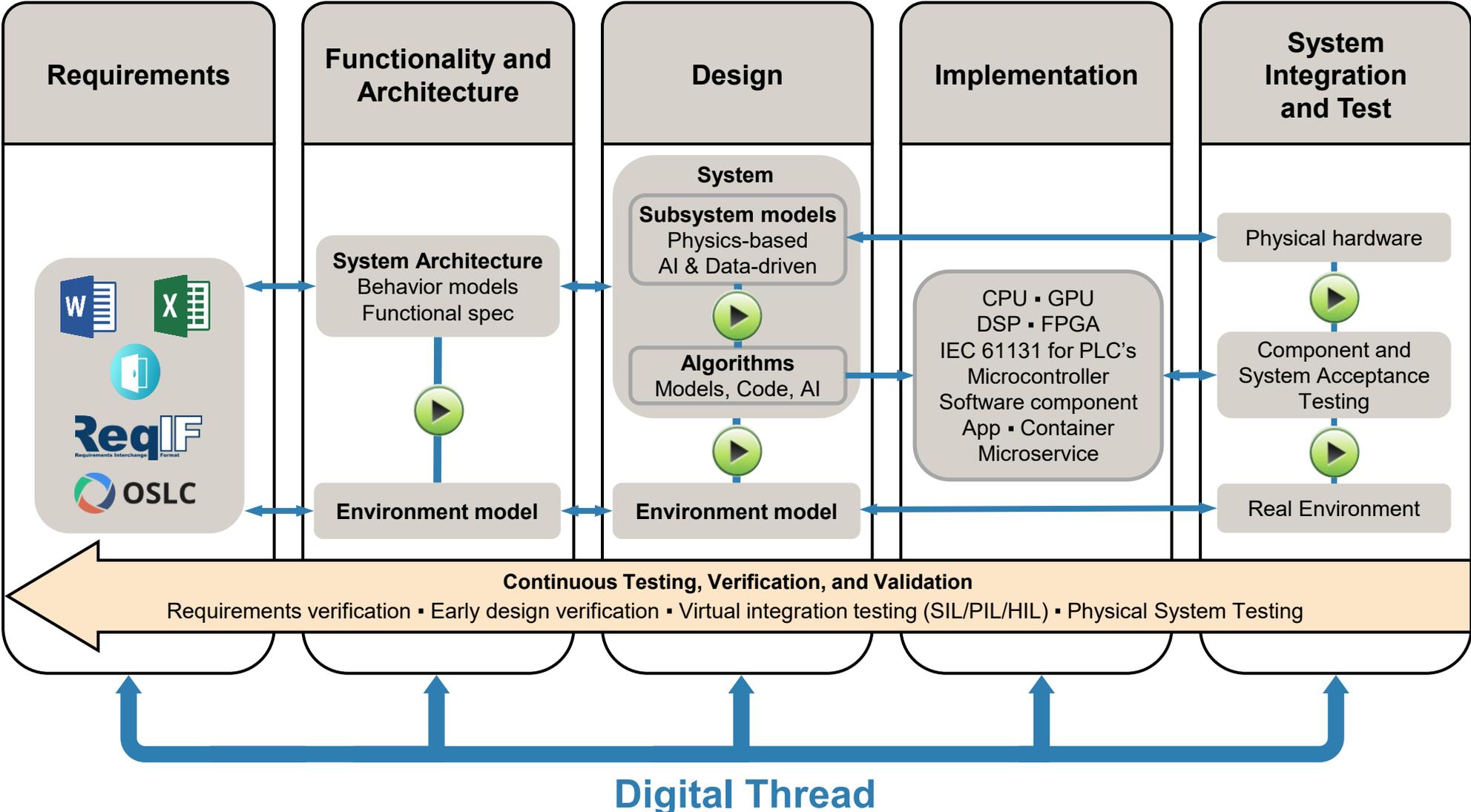
Agenda

- I. Introduction to AI with Model-Based Design
- II. Deep dive of Virtual Sensor Modeling workflow
- III. Resources for further learning

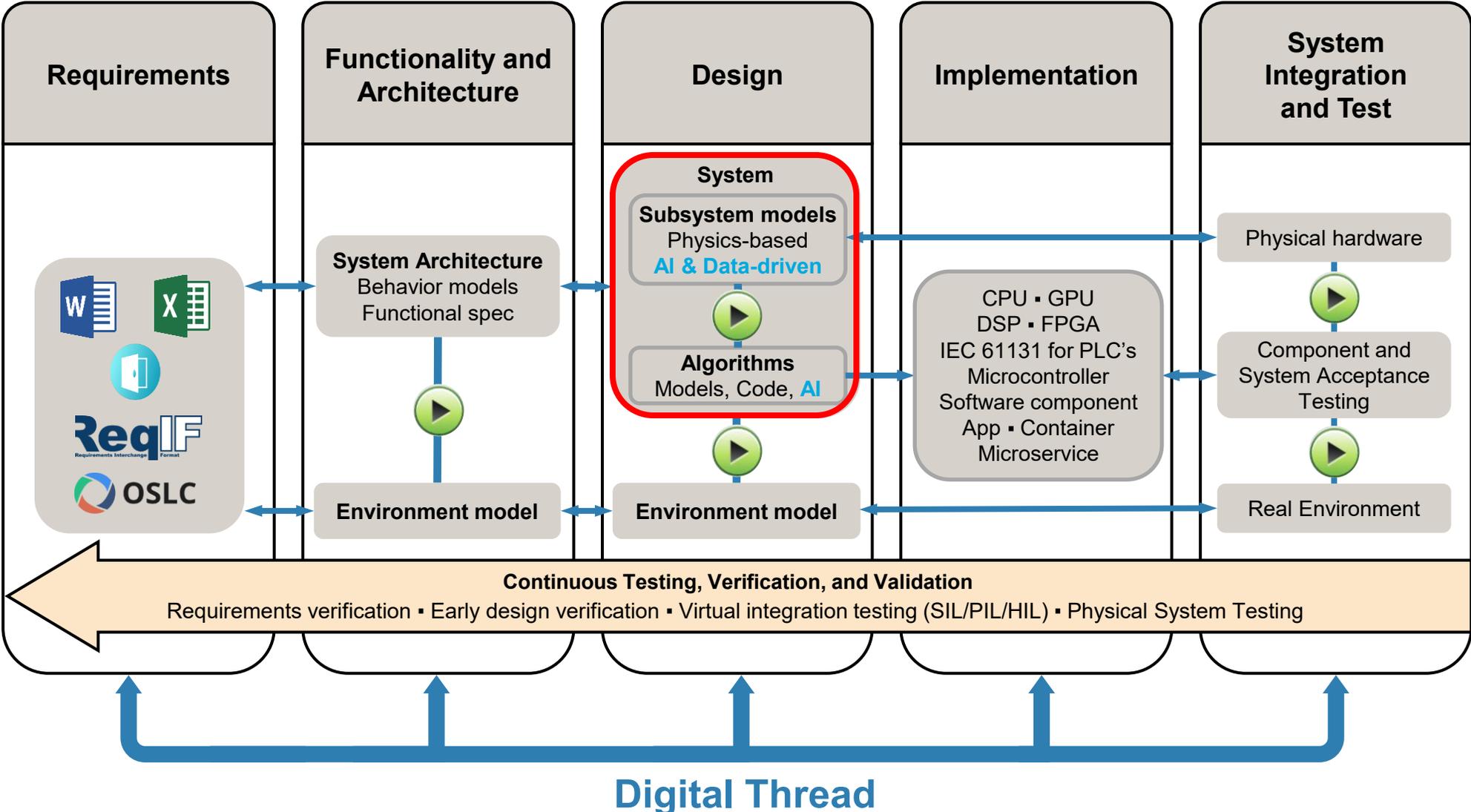
Agenda

- I. Introduction to AI with Model-Based Design
- II. Deep dive of Virtual Sensor Modeling workflow
- III. Resources for further learning

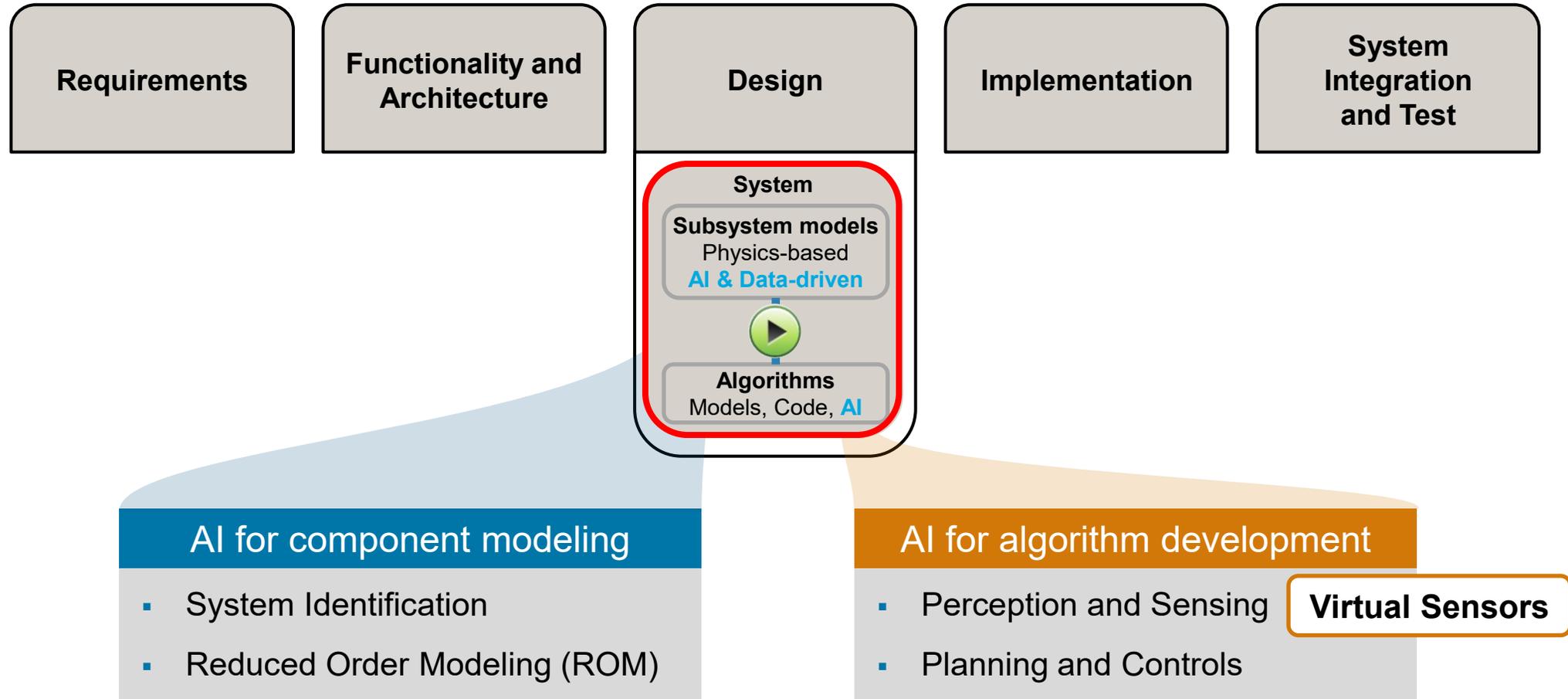
Model-Based Design



Integrating AI into Model-Based Design

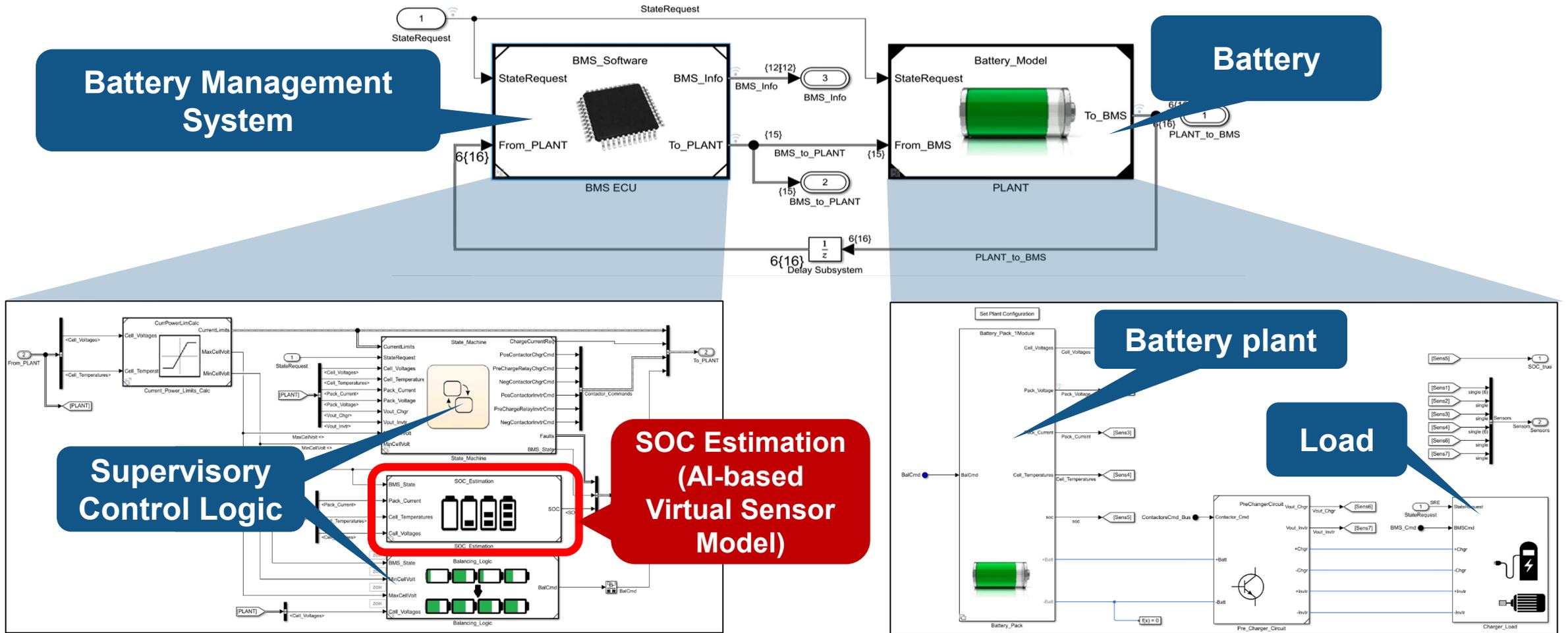


AI for Component Modeling and Algorithm Development



AI is often part of a larger system

Simulate and test all your components together in Simulink



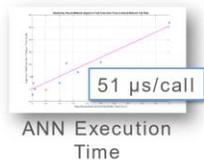
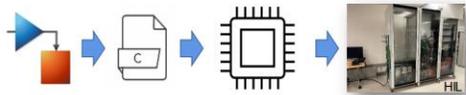
Virtual Sensor User Stories



Mercedes-Benz

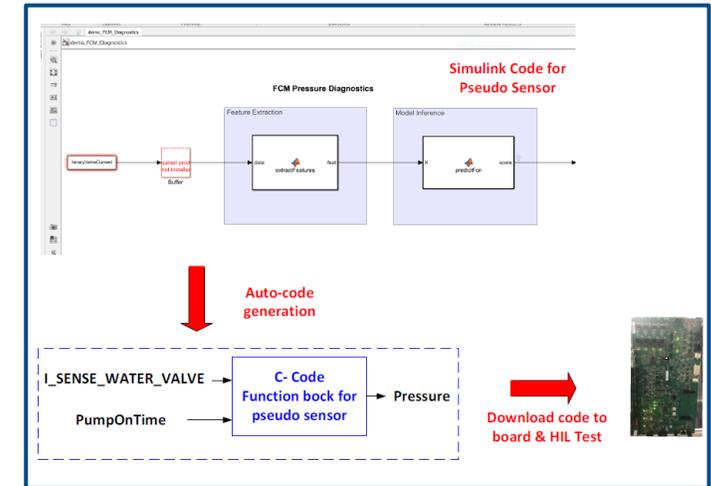
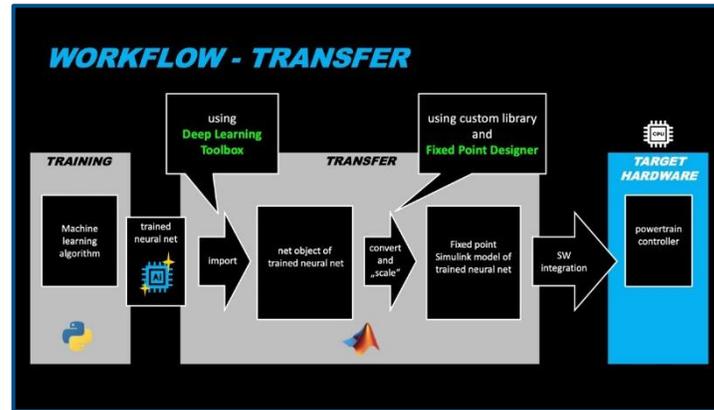


System Test on HIL



< 2 kB ROM
< 100 B RAM
ANN Memory Usage

1.98e-04
Mean Absolute Error
Numerical Equivalence
(Windows \leftrightarrow Microcontroller)



Gotion:

Battery Pack SOC Virtual Sensor with a Neural Network

Mercedes-Benz:

Engine Piston Virtual Pressure Sensors with Deep Neural Networks

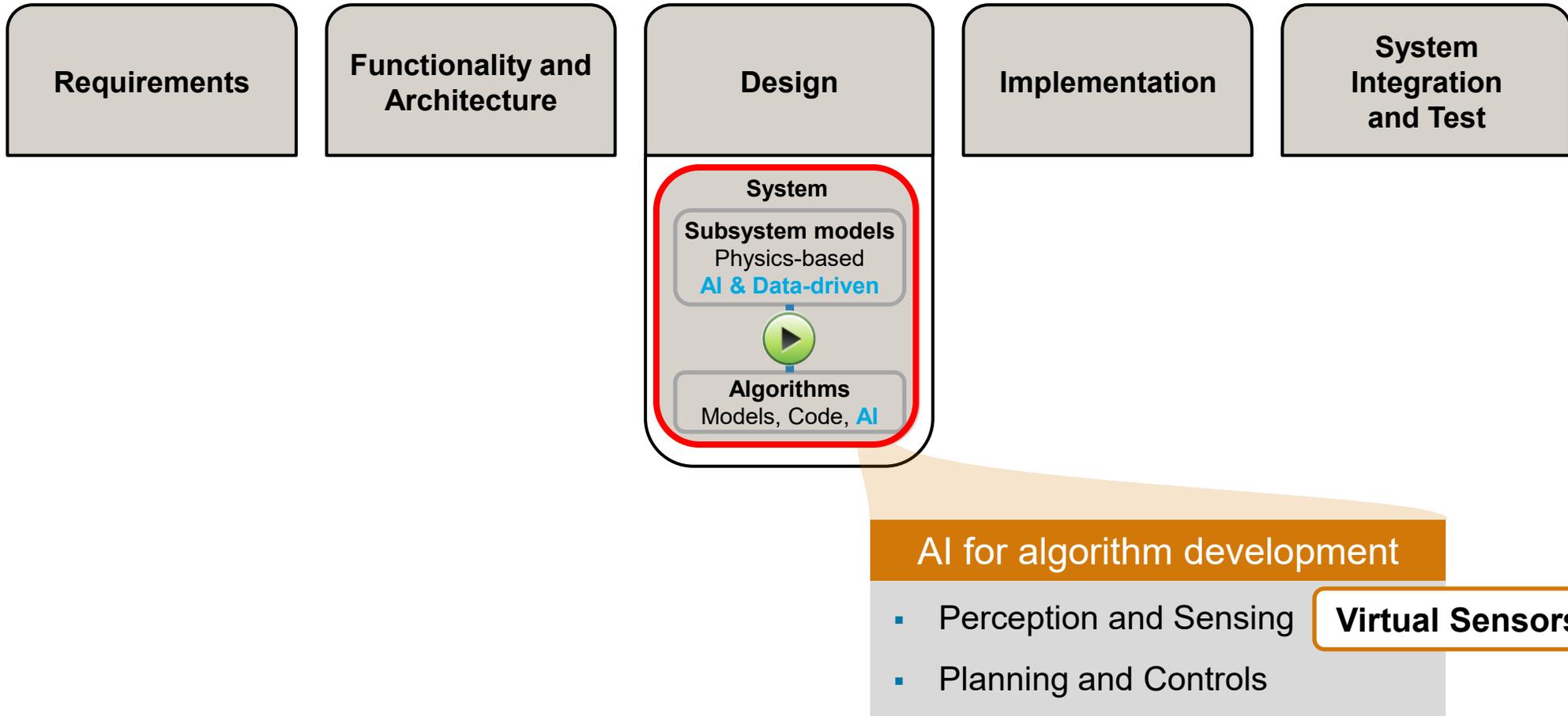
Coca-Cola:

Beverage Dispenser Virtual Pressure Sensor with Machine Learning

Agenda

- I. Introduction to AI with Model-Based Design
- II. Deep dive of Virtual Sensor Modeling workflow
- III. Resources for further learning

Focus today



A Virtual Sensor mimics a physical sensor using data from other measurements to estimate the quantity of interest



Why are Virtual Sensors relevant?



A **physical sensor** may be:

- Expensive
- Slow
- Noisy
- Unreliable
- Not feasible
- Unmanufacturable
- Degrading over time
- Requiring redundancy
- etc.

Data-driven vs. first-principles modeling

Data-driven models and first-principles models can co-exist

DATA-DRIVEN MODELS

Statistics, optimization, AI

FIRST-PRINCIPLES MODELS

Physics, math, domain knowledge

BLACK BOX

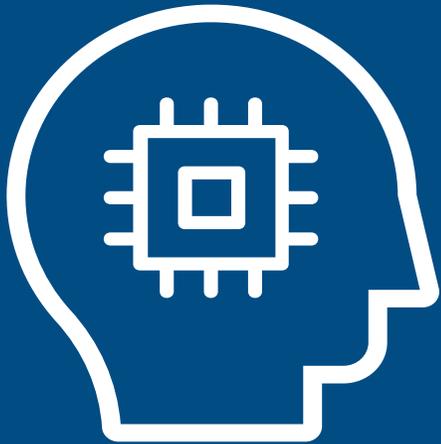
GREY BOX

WHITE BOX

The AI megatrend

ARTIFICIAL INTELLIGENCE

Any technique that enables machines to mimic human intelligence

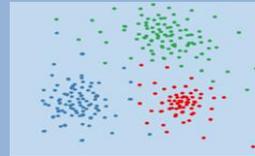


MACHINE LEARNING

Statistical methods that enable machines to “learn” tasks from data without explicitly programming

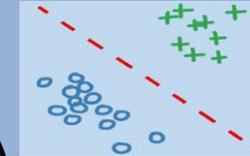
UNSUPERVISED LEARNING

(No Labeled Data)

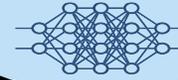


SUPERVISED LEARNING

(Labeled Data)

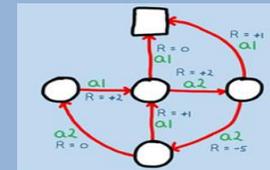


DEEP LEARNING
(Neural networks with many layers)

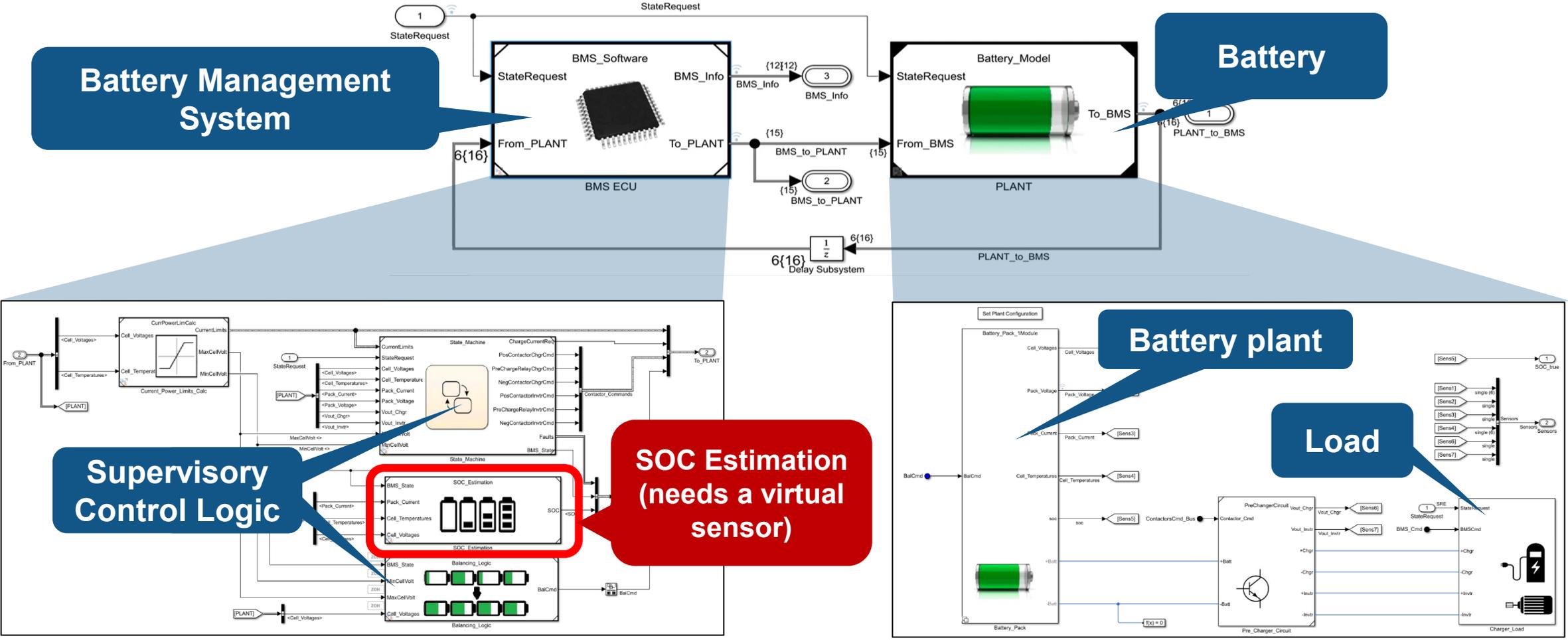


REINFORCEMENT LEARNING

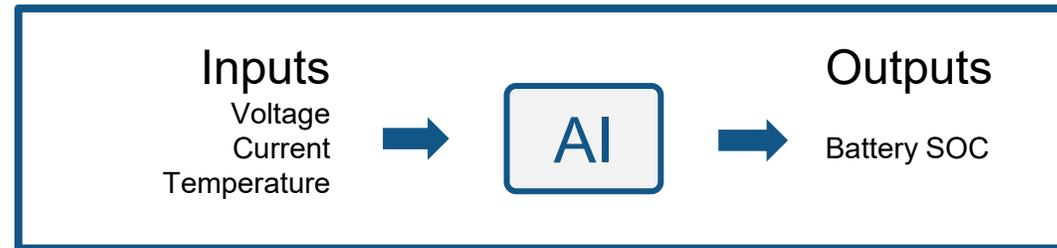
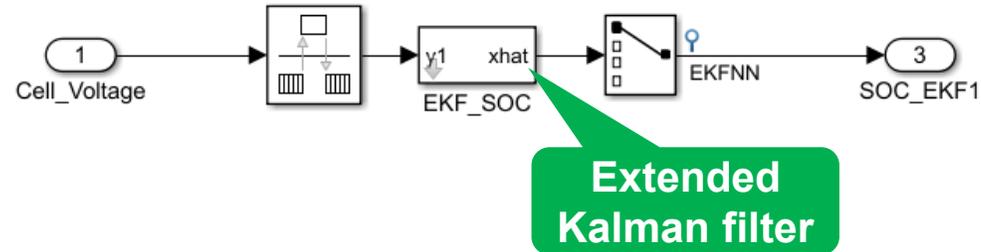
(Interaction Data)



Having a physical sensor might not be feasible due to cost, manufacturing process, reliability, degradation, etc.



Virtual sensor for Battery State of Charge (SOC) estimation



Feature	Kalman Filters	AI Methods
Nonlinearity	Requires EKF/UKF	Handles well
Data Requirements	Accurate model needed	Large datasets required
Proven History	Long history of reliable performance	Emerging and rapidly evolving

Battery State-of-Charge Estimation Using AI



Workflow for developing AI-based virtual sensors



Data Preparation

Toolboxes for **domain-specific** pre/post-processing

AI Modeling

Low-code AI Modeling and training through MATLAB Apps

Model import from **PyTorch, TensorFlow, and ONNX**

Simulation & Test

Simulink blocks for AI inference make integration easy

Simulation-based unit testing

Compression

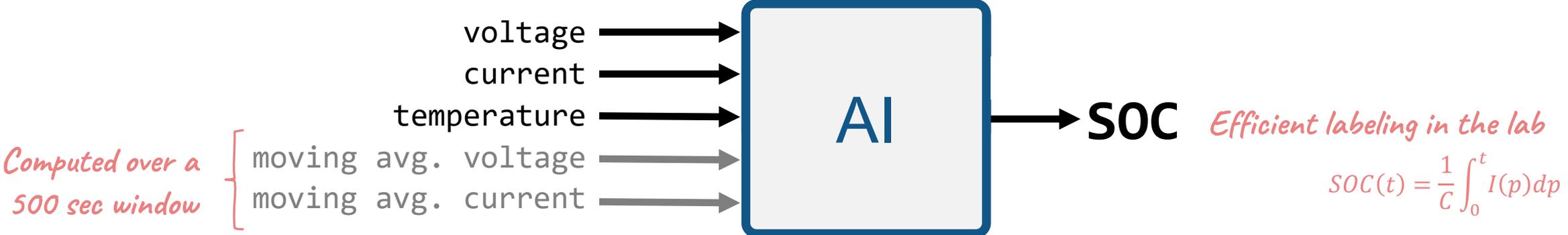
Model compression techniques to reduce model size and accelerate inference

Deployment

Automatic code generation for embedded targets

Data Preparation

Data source: McMaster University*



Predictors

Response

	1	2	3	4	5	6
	Voltage	Current	Temperature	Moving Average Voltage	Moving Average Current	SOC
1	0.7510	0.3851	0.3031	0.7510	0.3851	0.2064
2	0.7510	0.3852	0.3046	0.7510	0.3851	0.2064
3	0.7510	0.3852	0.3061	0.7510	0.3852	0.2064
4	0.7510	0.3852	0.3076	0.7510	0.3852	0.2064
5	0.7510	0.3852	0.3091	0.7510	0.3852	0.2064
6	0.7510	0.3852	0.3106	0.7510	0.3852	0.2064
7	0.7510	0.3852	0.3120	0.7510	0.3852	0.2064
8	0.7510	0.3852	0.3135	0.7510	0.3852	0.2064
9	0.7510	0.3852	0.3150	0.7510	0.3852	0.2064
10	0.7510	0.3852	0.3165	0.7510	0.3852	0.2064

* <https://data.mendeley.com/datasets/cp3473x7xv/3>

AI Modeling – emerging and rapidly evolving

Type	Classic machine learning	Deep Learning		Hybrid Approach
		Feedforward neural network (FNN)	Recurrent neural network (RNN)	
Description	Learn patterns from data; often require feature engineering	Data flows in one direction through layers	Designed for sequential data, captures temporal dependencies	Combines AI and traditional methods, leverages the strengths of both approaches
Algorithms	Decision Trees, Random Forests, SVMs, etc.	FNN (also known as FFN)	Long-Short Term Memory (LSTM); Gated Recurrent Unit (GRU)	Kalman Filter + AI

AI Modeling – emerging and rapidly evolving

Type	Classic machine learning	Deep Learning	
		Feedforward neural network (FNN)	Recurrent neural network (RNN)
Description	Learn patterns from data; often require feature engineering	Data flows in one direction through layers	Designed for sequential data, captures temporal dependencies
Algorithms	Decision Trees, Random Forests, SVMs, etc.	FNN (also known as FFN)	Long-Short Term Memory (LSTM); Gated Recurrent Unit (GRU)

1
Train in MATLAB's **Machine Learning** Framework

2
Train in MATLAB's **Deep Learning** Framework

3
Import model from **TensorFlow or PyTorch**

Train a regression tree model

1

Train in MATLAB's **Machine Learning Framework**

Battery State-of-Charge (SOC) Estimation using Machine Learning

In this example, you will create an AI model for performing battery State-of-Charge (SOC) estimation by loading data, training and evaluating regression models using Machine Learning via low-code techniques.

Table of Contents

1. Data Preparation
 - 1.1. Load training and validation data
 - 1.2. Visually explore the data
2. Overview of the AI Model
3. Train a machine learning model
 - 3.1. Train and compare different models interactively using Regression Learner app
 - 3.2. Train a specific Machine Learning model on the command line
4. Test your Machine Learning model at various Temperature ranges

1. Data Preparation

The training data contains a single sequence of experimental data collected while the battery powered an electric vehicle during four driving cycles at different temperatures (-10°C, 0°C, 10°C and 25°C). The test data contains four sequences of experimental data collected during driving cycles at the same temperatures.

All the data required for this exercise has been downloaded and is available in the data folder. [Learn more](#) about the dataset.

Data is stored in three different folders: Train, Validation and Test, with variables saved in MAT files. Input variable is a (*features x observations*) matrix "X" and the response variable is a row vector Y.

```
1 dataFolder = fullfile("data", "LGHG2@n10C_to_25degC");
```

1.1. Load training and validation data

Command Window

```
>>
```

Project Issues

Startup (0) Checks (0) Shutdown ...

No issues to display. Issues that occur when opening a project appear here.

[Learn more about startup issues](#)

Editor: 100% UTF-8 LF Script

Data Preparation

AI Modeling

Simulation & Test

Deployment

Train a feedforward network network

2

Train in MATLAB's **Deep Learning Framework**

Battery State-of-Charge (SOC) Estimation using a Feed-Forward Network

In this example, you will create an AI model for performing battery State-of-Charge (SOC) estimation by loading data, defining the architecture of a feed-forward neural network, training the network, and exporting the trained network to Simulink for running simulations.

Table of Contents

1. Data Preparation
 - 1.1. Load training and validation data
 - 1.2. Visually explore the data
2. Overview of the AI Model
3. Train a feedforward neural network (FFN)
 - 3.1. Layout network architecture
 - 3.2. Export the network architecture to the MATLAB workspace
 - 3.3 Set Training Options and train the network
4. Test your Feedforward Neural Network at various Temperature ranges

1. Data Preparation

The training data contains a single sequence of experimental data collected while the battery powered an electric vehicle during four driving cycles at different temperatures (-10°C, 0°C, 10°C and 25°C). The test data contains four sequences of experimental data collected during driving cycles at the same temperatures.

All the data required for this exercise has been downloaded and is available in the data folder. [Learn more](#) about the dataset.

Data is stored in three different folders: Train, Validation and Test, with variables saved in MAT files. Input variable is a (*features x observations*) matrix 'X' and the response variable is a row vector Y.

```
dataFolder = fullfile('data', 'LGHG2@n10C_to_25denC');
```

Data Preparation

AI Modeling

Simulation & Test

Deployment

Train an LSTM network

2

Train in MATLAB's **Deep Learning Framework**

The screenshot displays the MATLAB R2025a environment. The main window shows a document titled "SOC_2_DeepLearning_LSTM.mlx" with the following content:

Battery State-of-Charge Estimation using Deep Learning (LSTM)

In this example, you will create an AI model for performing battery State-of-Charge (SOC) estimation by loading data, defining the architecture of a Long Short-Term Memory network (LSTM), training the network, and exporting the trained network to Simulink for running simulations.

Table of Contents

1. Data Preparation
 - 1.1. Load training and validation data
2. Overview of the AI Model
3. Train a long-short term memory network (LSTM)
 - 3.1 Define Network Architecture
 - 3.2 Set Training Options and train the network
3. Test the Network

1. Data Preparation

The training data contains a single sequence of experimental data collected while the battery powered an electric vehicle during four driving cycles at different temperatures (-10°C, 0°C, 10°C and 25°C). The test data contains four sequences of experimental data collected during driving cycles at the same temperatures.

All the data required for this exercise has been downloaded and is available in the data folder. [Learn more](#) about the dataset.

Data is stored in three different folders: Train, Validation and Test, with variables saved in MAT files. Input variable is a (*features x observations*) matrix 'X' and the response variable is a row vector Y.

```
1 dataFolder = fullfile('data', 'LGHG2@n10C_to_25degC');
```

Command Window

```
>> Press [Ctrl]P to generate code with Copilot
```

On the right side, the Copilot Chat window is open, displaying a welcome message and several example prompts:

- Remember that Copilot sometimes writes code and text that seems accurate, but is not. Make sure to verify any received code and give feedback on the results to help improve the responses.
- Shuffle Example Prompts
- Roll two six-sided dice 1000 times and plot the sum of each roll
- Solve the linear equations with coefficients A = [2, 4; 1, 3] and constants B = [8; 5] and display the results
- Load Fisher's iris data set, use the petal lengths and widths as predictors, cluster data using k-means clustering, and then plot the cluster regions

At the bottom right, there is an "Ask Copilot" button and a note: "Validate generated output before use."

Data Preparation

AI Modeling

Simulation & Test

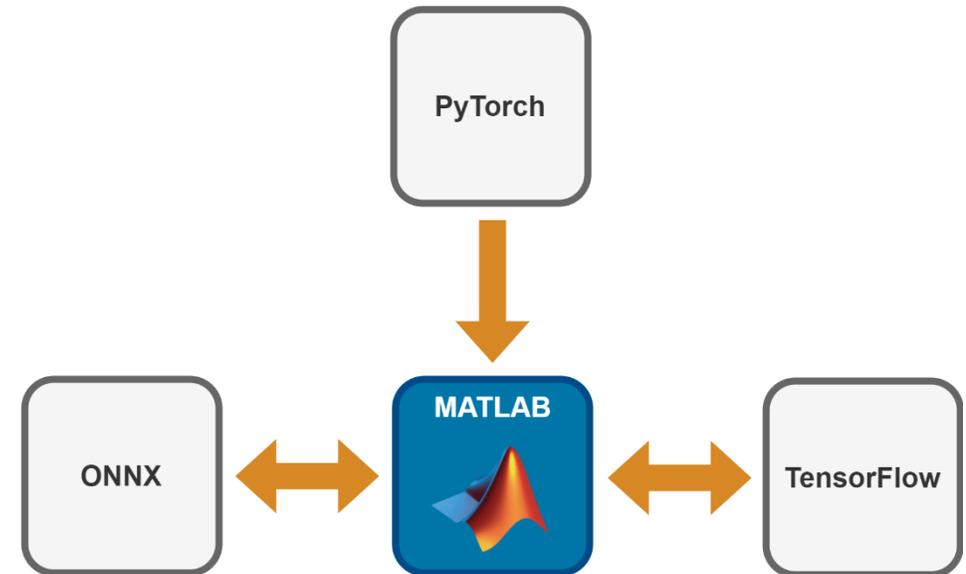
Deployment

Interoperate with Open-Source Frameworks

Framework Interoperability bridges the gap between data science, engineering and production

Import and Export models through:

TensorFlow-Keras Import	R2017b
ONNX Converter (Import & Export)	R2018a
TensorFlow Converter (Import)	R2021a
TensorFlow Converter (Export)	R2022b
PyTorch Converter (Import)	R2022b
New functions for importing networks from ONNX and TensorFlow to dlnetwork	R2023b
Import networks from PyTorch and TensorFlow in Deep Network Designer	R2023b



Import a trained PyTorch model

3

Import model from
TensorFlow or PyTorch

Battery State-of-Charge (SOC) Estimation using an imported PyTorch model

In this example, you will use an AI model built in PyTorch for performing battery State-of-Charge (SOC) estimation by importing the trained PyTorch model into MATLAB.

Table of Contents

1. Import PyTorch Network into MATLAB
(Recommended) Option 1: Use Deep Network Designer
Option 2: Import programmatically
2. Test your imported PyTorch Network at various temperature ranges

1. Import PyTorch Network into MATLAB

Import a pretrained and traced PyTorch model as an uninitialized `dlnetwork` object. Then, add an input layer to the imported network.

Import a network from an external platform using Deep Network Designer, or programmatically.

(Recommended) Option 1: Use Deep Network Designer

Open **Deep Network Designer app** from the toolbar,

Command Window
>>

Copilot Chat
Hello! Copilot is here to answer your questions, help write and explain code, and even identify code issues. Learn more about [Copilot](#).

Remember that Copilot sometimes writes code and text that seems accurate, but is not. Make sure to verify any received code and give feedback on the results to help improve the responses.

Shuffle Example Prompts

- Roll two six-sided dice 1000 times and plot the sum of each roll
- Solve the linear equations with coefficients $A = [2, 4; 1, 3]$ and constants $B = [8; 5]$ and display the results
- Load Fisher's iris data set, use the petal lengths and widths as predictors, cluster data using k-means clustering, and then plot the cluster regions

Ask Copilot

Validate generated output before use.

Editor: 100% UTF-8 LF Script

Data Preparation

AI Modeling

Simulation & Test

Deployment

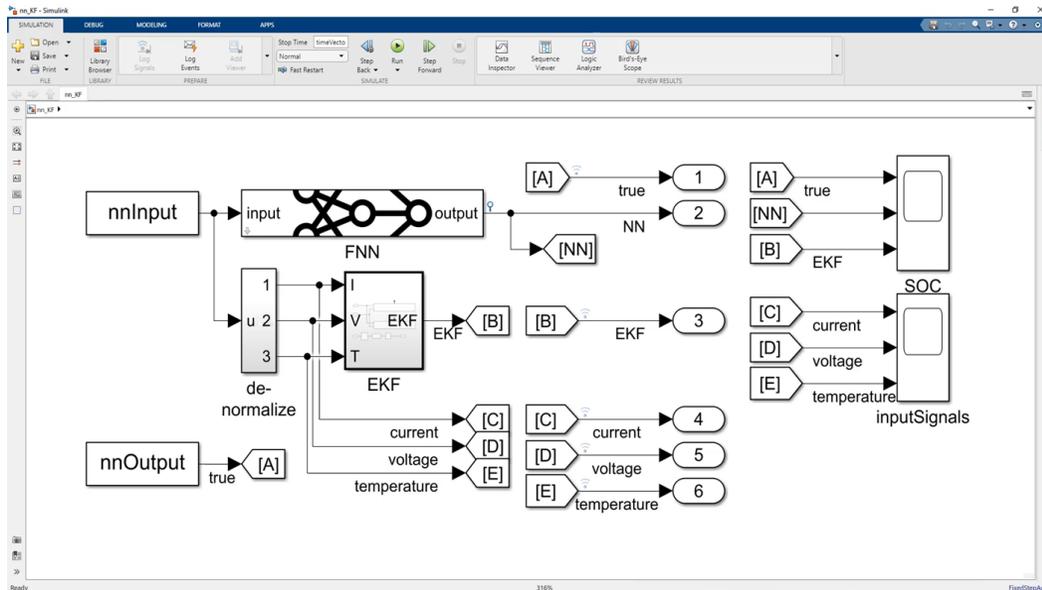
AI Modeling is an iterative process

Each method has its own strengths and weaknesses. The choice often depends on the specific requirements and constraints of the application.

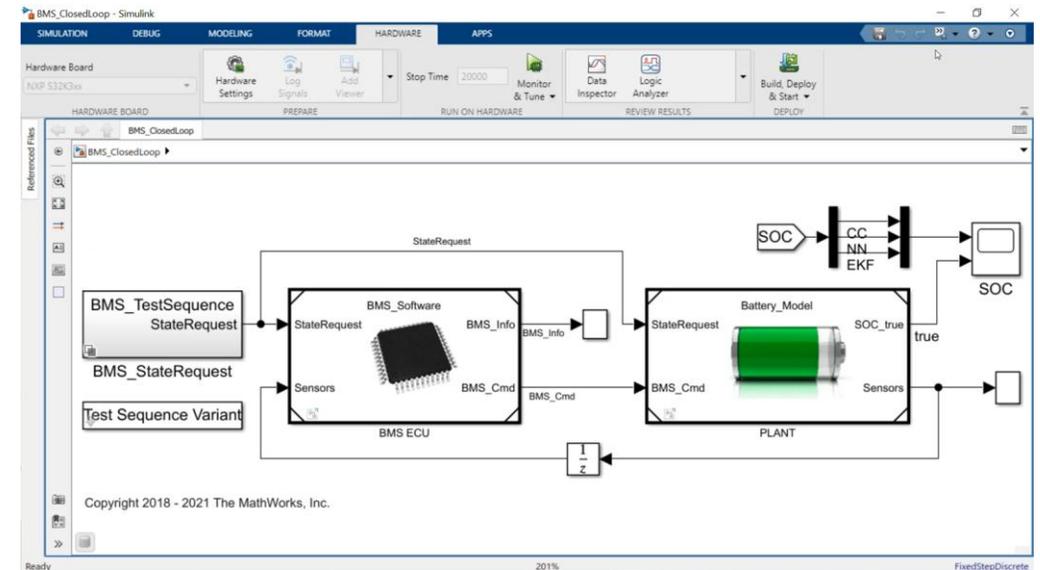
	Regression Tree	FFN	LSTM	GRU	Nonlinear ARX	Kalman Filter
Benefits	Simplicity and interpretability	Small model size, scales well with data, fast for inference	Suitable for sequence data, captures nonlinear dynamics well	Efficient with fewer parameters and run faster than LSTM	Provide physical insights, flexible choices of nonlinear estimators including AI	Real-time efficiency, can be combined with AI models to improve accuracy
Challenges	Feature engineering needed	Feature calculation and hyperparameter tuning needed	Computationally intensive, hyperparameter tuning needed	May not capture long dependencies as well as LSTM	Trial-and-error for the choice of nonlinear estimators	Highly dependent on the battery model, sensitive to system noise

Integrate your AI model for system-level simulation and test

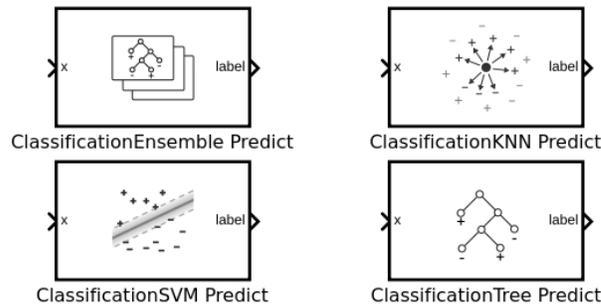
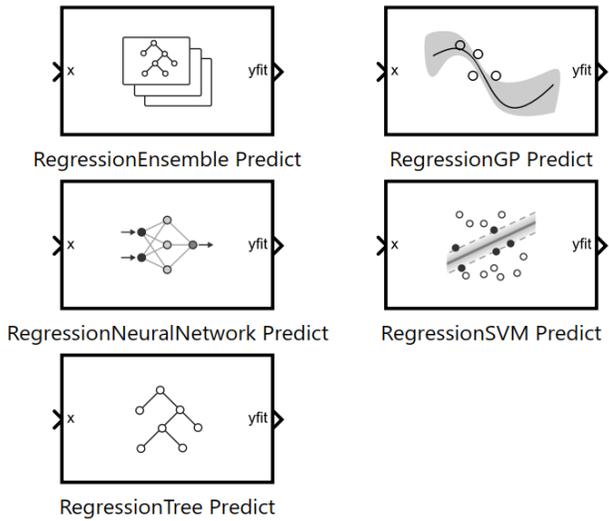
Integration of trained AI model into Simulink



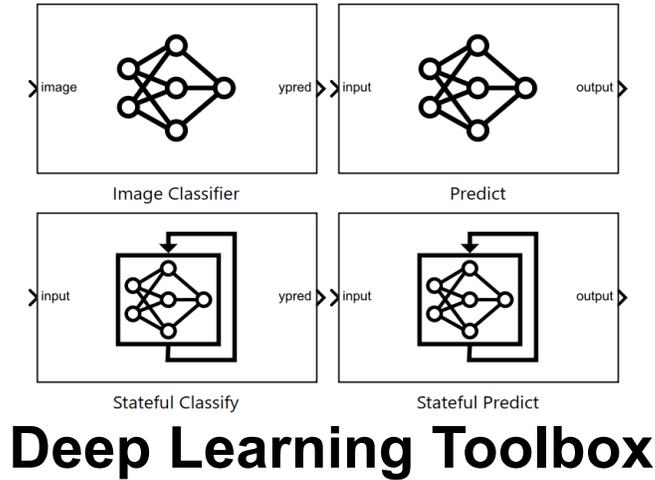
System-level simulation



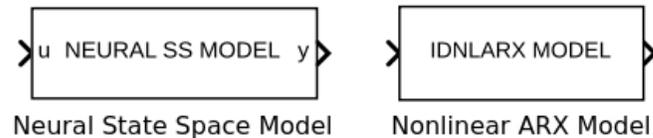
AI libraries in Simulink are expanding to include more AI blocks for more applications



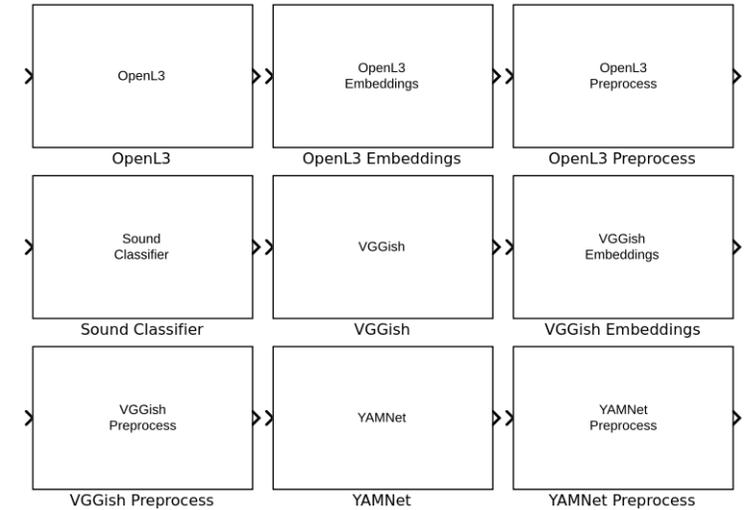
Statistics and Machine Learning Toolbox



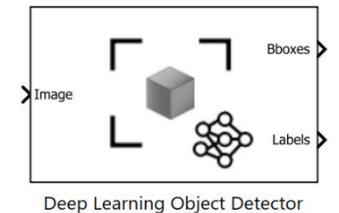
Deep Learning Toolbox



System Identification Toolbox



Audio Toolbox



Computer Vision Toolbox

Data Preparation

AI Modeling

Simulation & Test

Deployment

Integration of trained AI models into Simulink

The screenshot displays the MATLAB R2025a Live Editor interface. The main window shows a script titled "Integrating AI models into Simulink". The script's content is as follows:

Integrating AI models into Simulink

This script intends to integrate the AI models trained in the previous section into Simulink. This allows us to compare each model's accuracy and proceed to system-level simulation and code generation.

Table of Contents

- Load test data set to use as inputs for Simulink simulation
- Define model inputs
- Integrate Machine Learning and Deep Learning Models into Simulink
- Select Model to Simulate
- Simulate model and evaluate performance
- Performance Comparisons

Load test data set to use as inputs for Simulink simulation

```
1 dataFolder = fullfile("data", "LHG2@n10C_to_25degC");
2 testFolder = fullfile(dataFolder, "Test");
3 testFiles = dir(testFolder+'/*.mat');
4
5 testTemperatureFile = 10°C; % 4 different choices of temperature profiles to load in
6 S = load(fullfile(testFolder, testTemperatureFile));
7 Xtest = S.X';
8 Ytest = S.Y';
```

Define model inputs

There are many ways to define input data for a Simulink model. You can use a root level inport, a From-workspace block, From File blocks and others. We are going to use From-workspace blocks.

We need to define a step size T_s for the model. We create a column vector `timeVector` of simulation time steps. Also, we define the inputs and outputs of the Simulink model.

```
9 Ts = 1;
10 tstop = size(Xtest,1)-1;
11 timeVector = (0:Ts:tstop)';
12 inputs = [timeVector, Xtest];
```

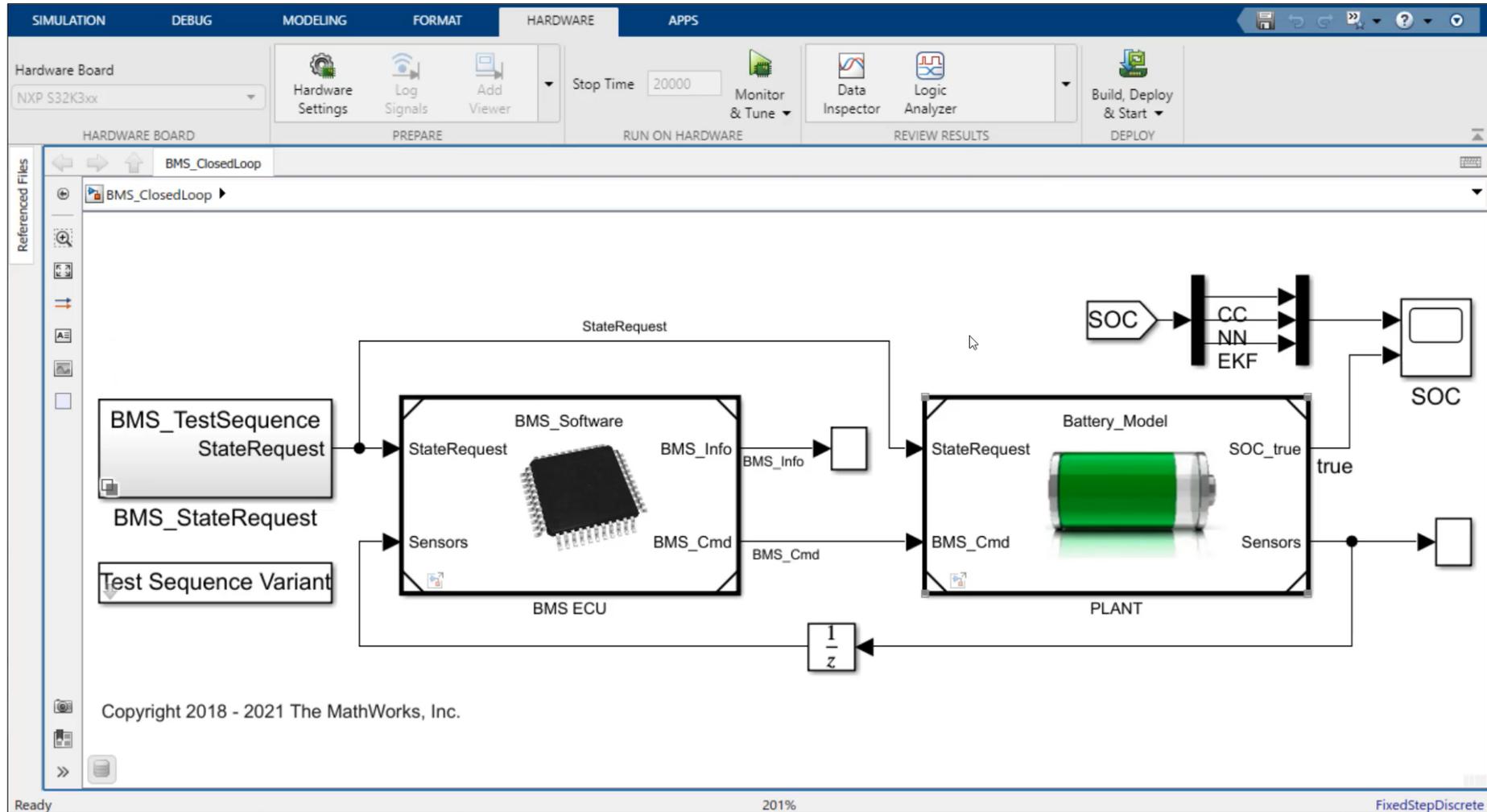
Data Preparation

AI Modeling

Simulation & Test

Deployment

Closed-Loop System-Level Simulation



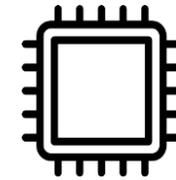
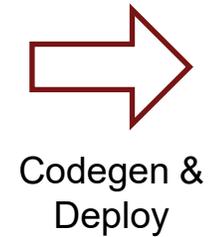
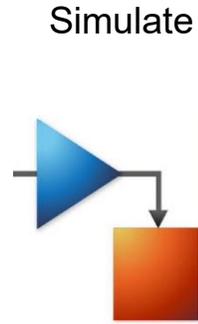
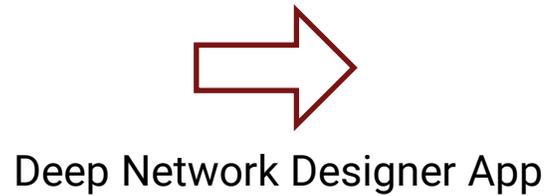
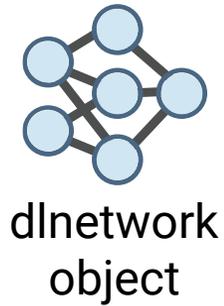
Data Preparation

AI Modeling

Simulation & Test

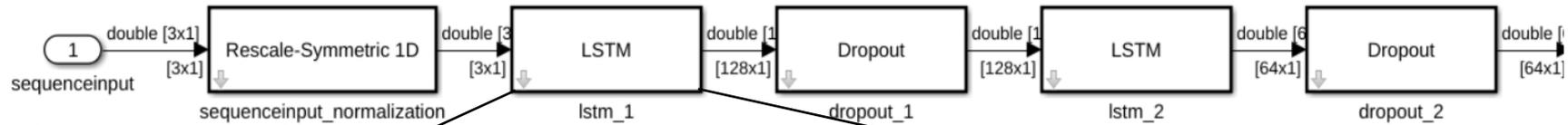
Deployment

Deep learning layer blocks enable layer-level network modeling in Simulink

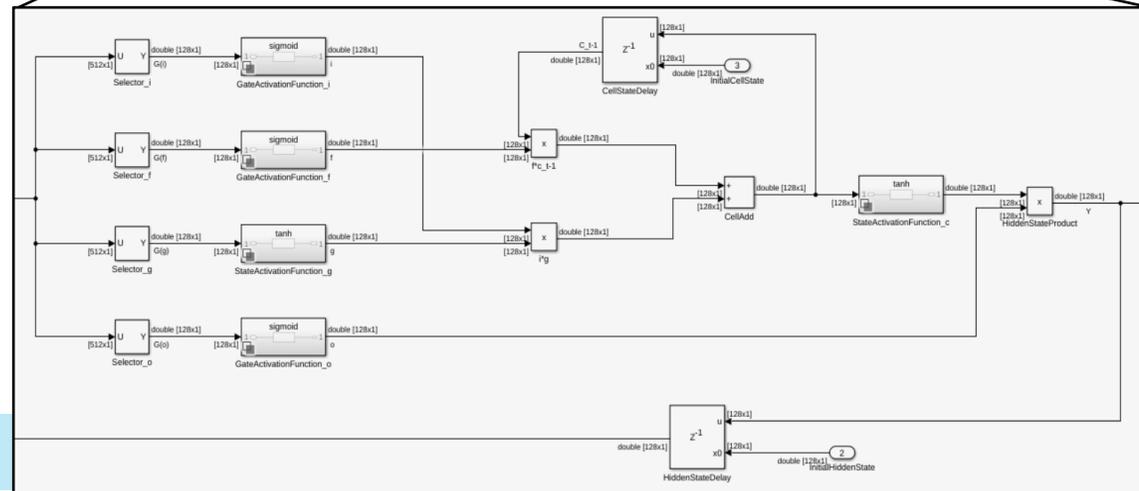


C/C++ Code Generation

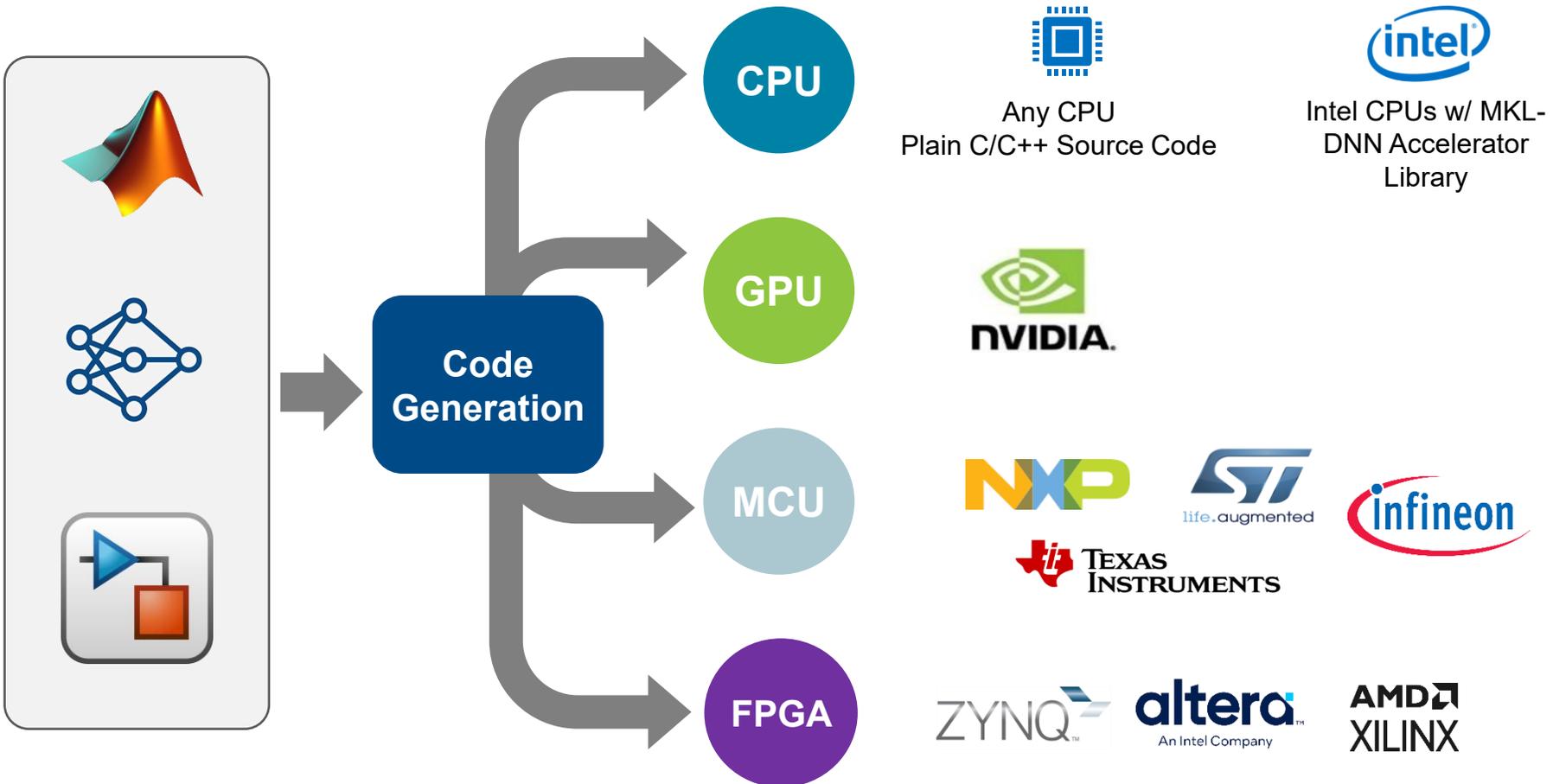
`exportNetworkToSimulink(net)`



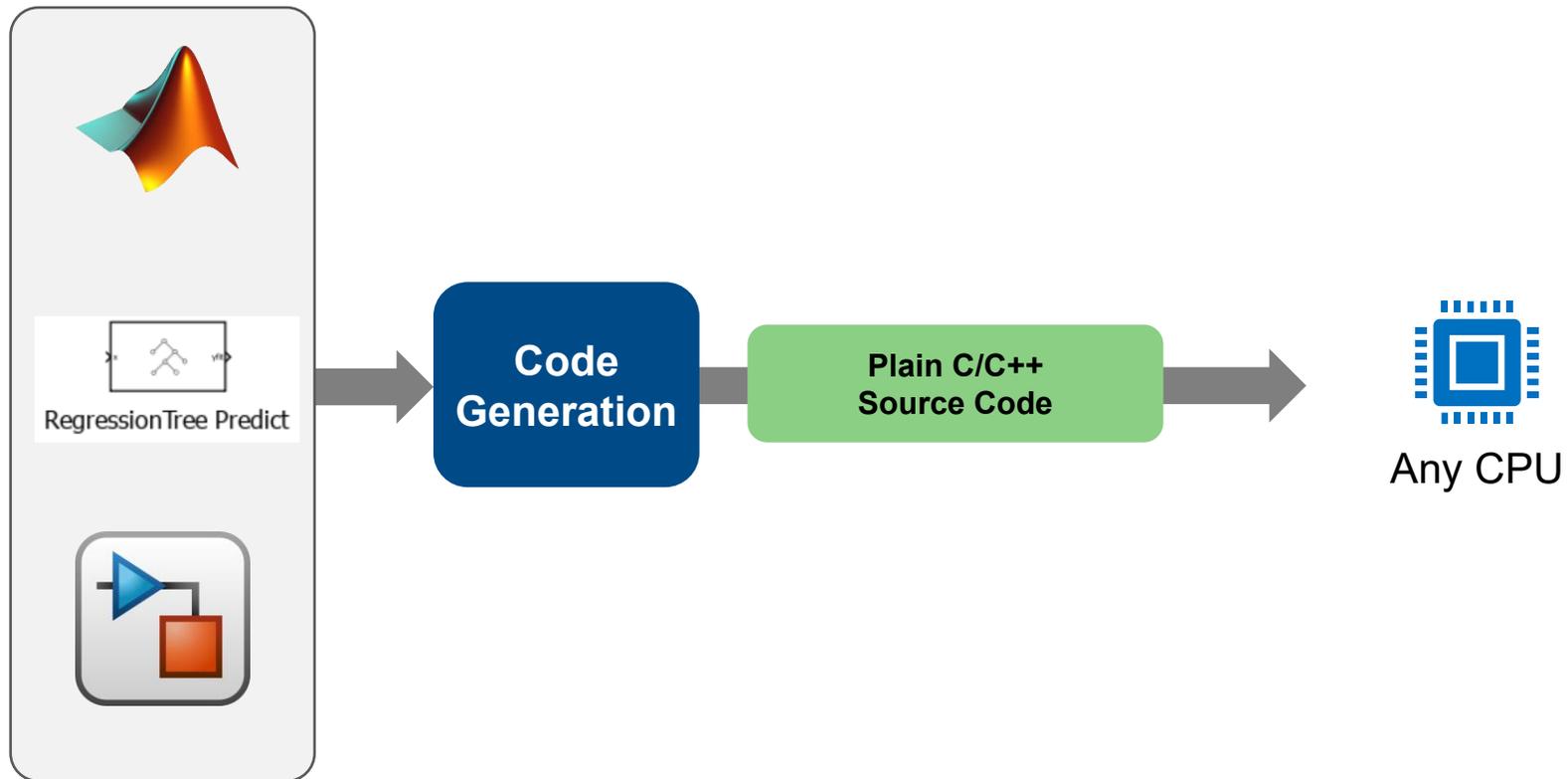
R2024b



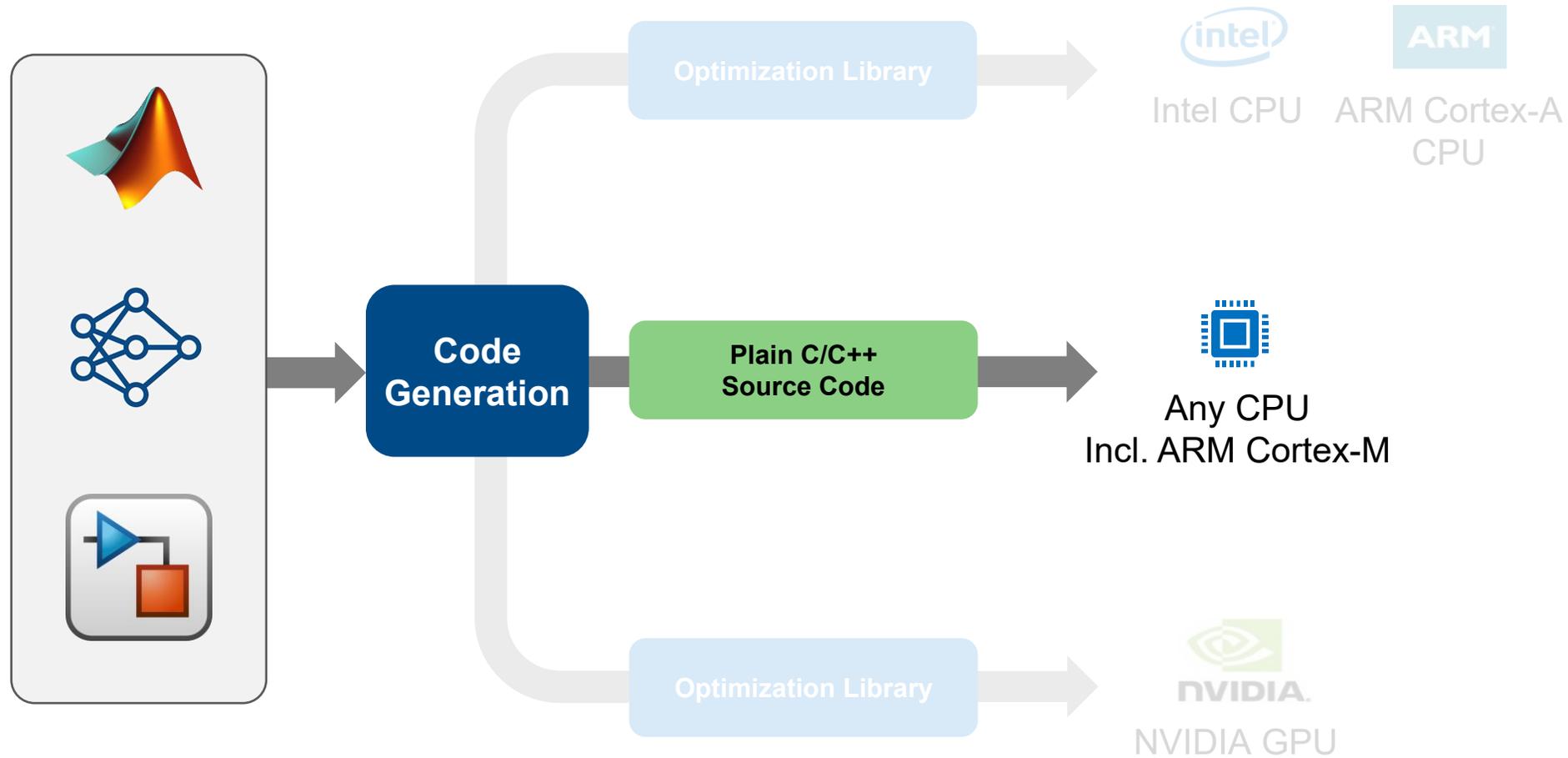
Deploy AI models to Host Machines and Embedded Hardware



Use Embedded Coder to Generate Code for Machine Learning



Generate Target Independent C/C++ Code for Deep Learning Networks



Data Preparation

AI Modeling

Simulation & Test

Deployment

Generate Code

The screenshot displays the MATLAB R2025a Live Editor interface. The title bar indicates the current file is 'Code Generation.mlx'. The main workspace contains a document titled 'Generate Code and Profile Performance' with the following text:

In this exercise, we will use Simulink to generate C code for one of the AI models we've trained so far. Then, we will profile its performance using Embedded Coder to measure its inference speed.

Table of Contents

- Load Model Inputs
- Select Model
- Generate Code
- Inspect Generated Code
- Measure Model Inference Speed

The document is divided into four numbered sections:

- Load Model Inputs**
Load the model inputs created in the previous exercise.

```
1 clear  
2 load("modelInputs.mat")
```
- Select Model**
Select a model to generate C code from.

```
3 model = Extended Kalman F...;
```
- Generate Code**
Open the model:

```
4 open_system("AI_models_codegen");
```

To generate code from the model, navigate to the **Apps** tab and open **Embedded Coder**.

The interface also shows a file explorer on the left with folders like 'AI_models_codegen_ert_rt' and 'slprj', and a workspace table at the bottom left.

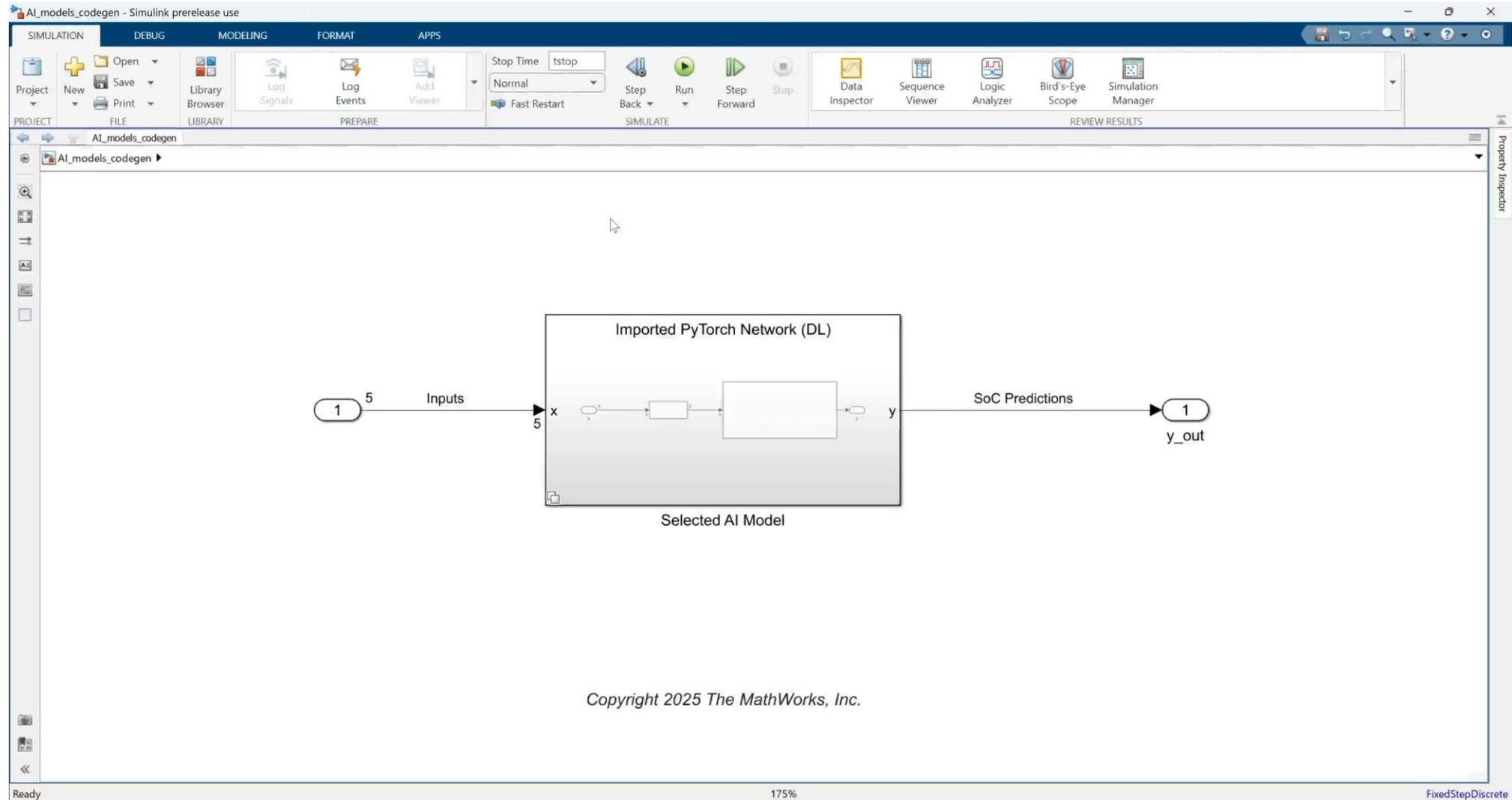
Data Preparation

AI Modeling

Simulation & Test

Deployment

Profile Performance



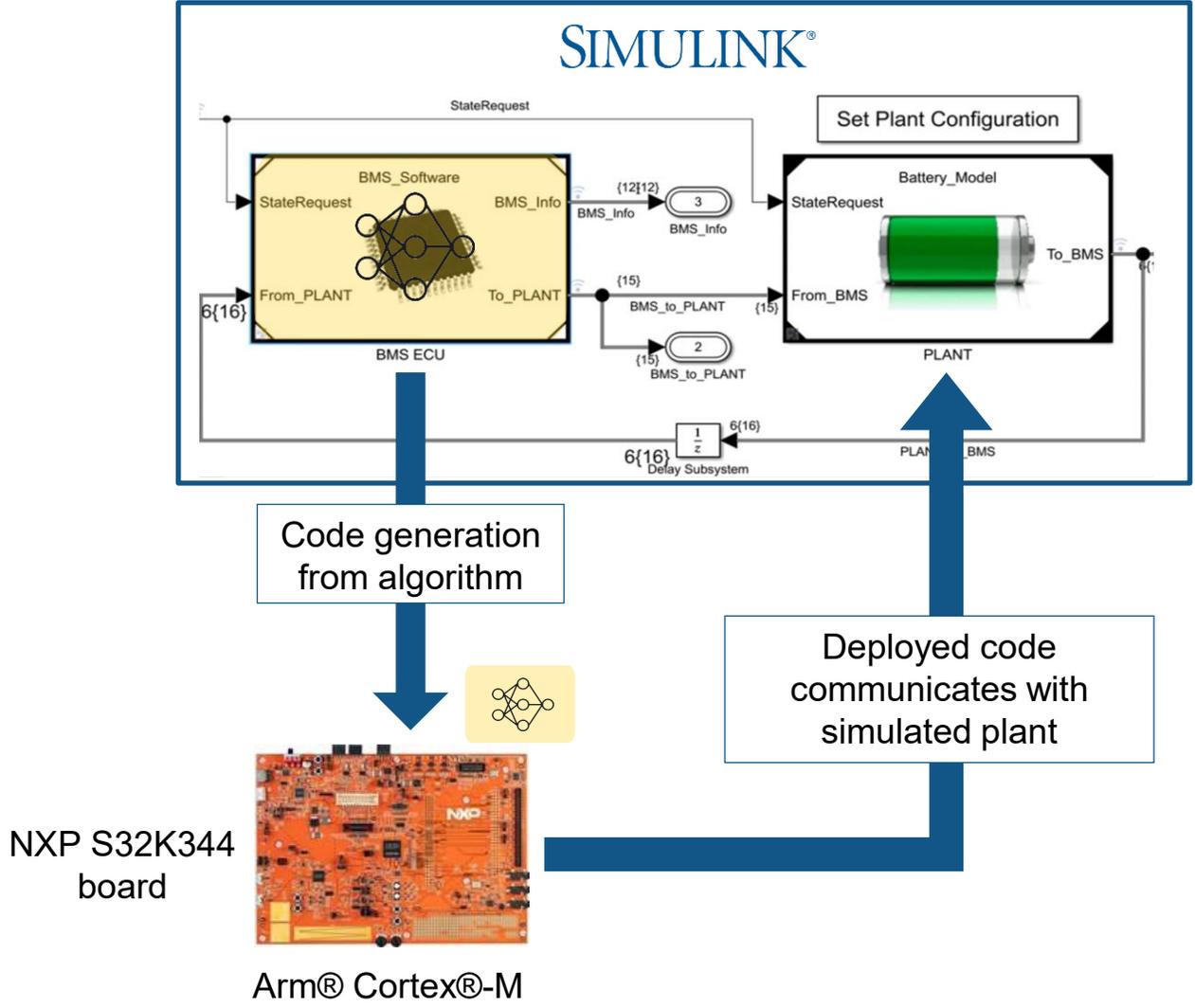
Data Preparation

AI Modeling

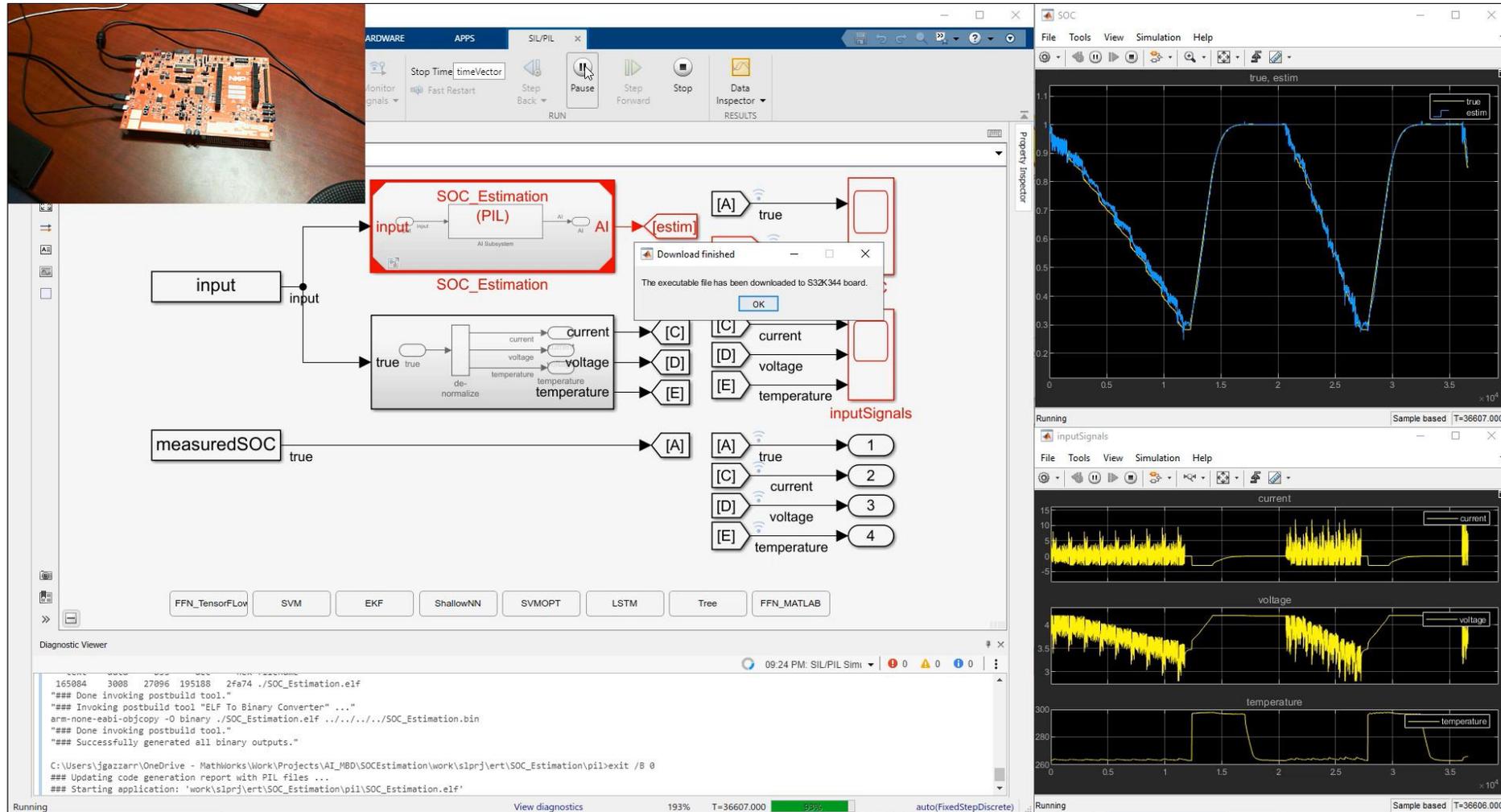
Simulation & Test

Deployment

Processor-in-the-Loop Testing on ARM Cortex-M7 Processor



Processor-in-the-Loop Testing on ARM Cortex-M7 Processor



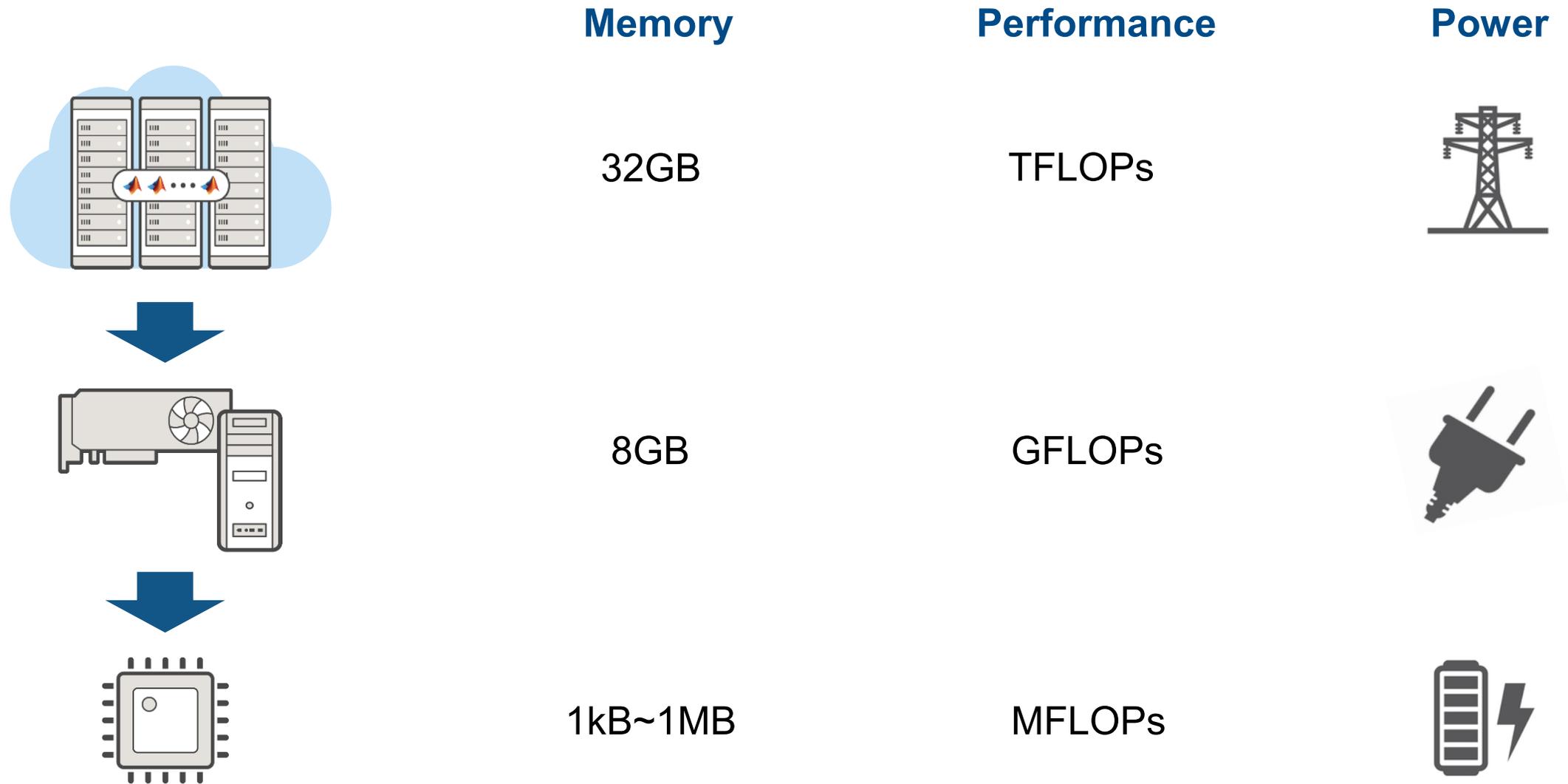
Data Preparation

AI Modeling

Simulation & Test

Deployment

Embedded Devices Have Limited Resources



Data Preparation

AI Modeling

Simulation & Test

Deployment

Manage AI tradeoffs for your system

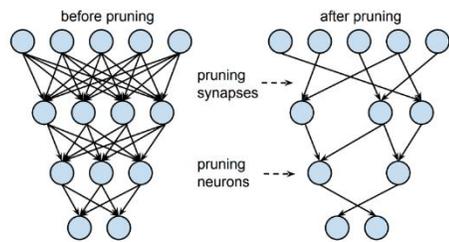
	EKF Extended Kalman Filter	Tree Fine Regression Tree	FFN 1-hidden layer Feedforward Network	LSTM Long Short-Term Memory Network
Preprocessing effort	●	●	●	●
Training Speed	N/A	●	●	●
Interpretability	●	●	●	●
Inference Speed	●	●	● ●	●
Model Size	●	●	●	●
Accuracy (RSME)	●	●	●	● ●

Results are specific to this Battery SOC Estimation example

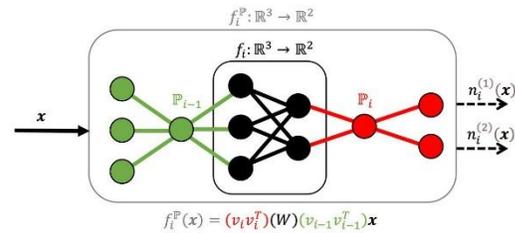


Deep learning model compression in MATLAB

Structural Compression (Deep Learning Models)



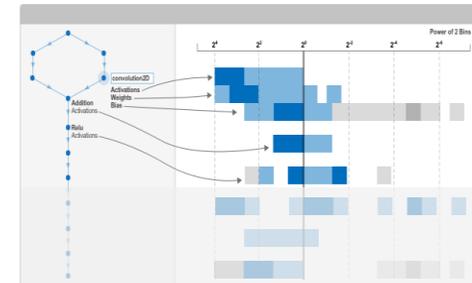
Pruning
convolutional neural
networks



Projection of deep
neural networks

We'll take a closer
look at the projection
technique in today's
seminar

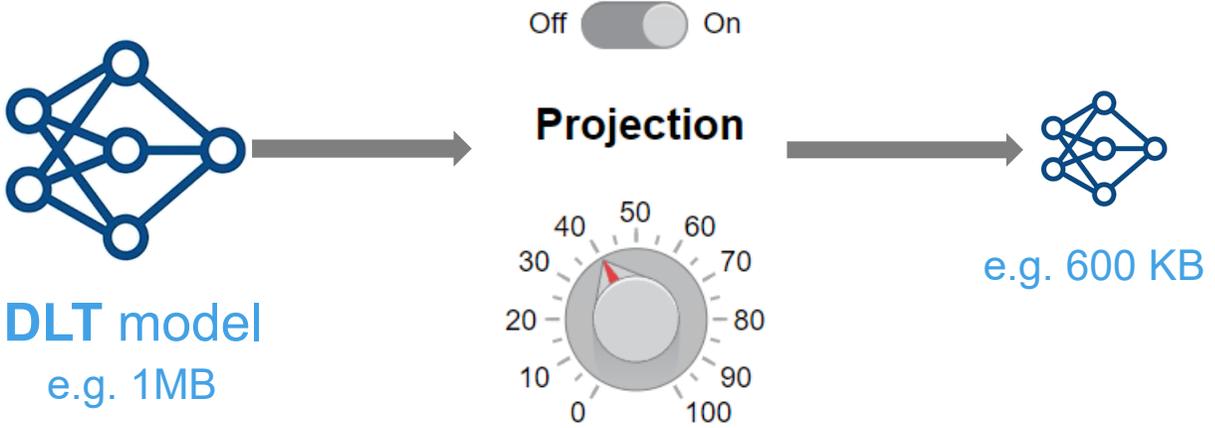
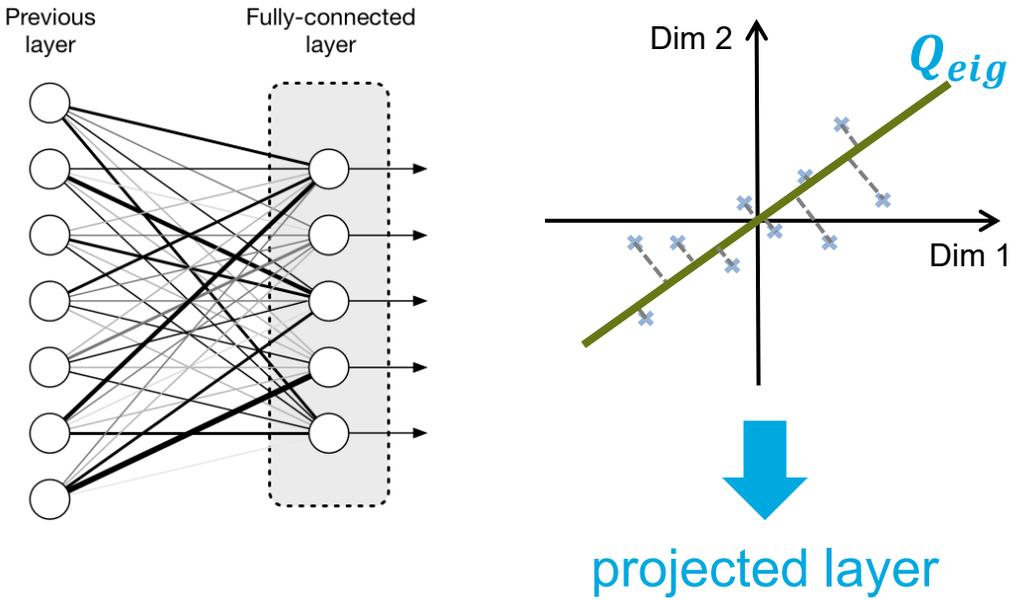
Datatype Compression (Machine Learning + Deep Learning Models)



Quantization of network
weights to lower precision
datatypes (bfloat16, int8)

Projection is a structural compression technique that can help reduce the size of the model while retaining model accuracy

Assumption: High-dimensional space of input and output neurons is redundant



Projection finds optimal subspaces with which to perform layer operations, and can be tuned to meet specific compression requirements

Compress LSTM Model with Projection

Model Compression

This section explores a novel method for compressing `dlnetwork` objects called [neural network projection](#). The technique can be applied to `dlnetwork` objects that are created and trained in MATLAB's Deep Learning Toolbox or imported from other Python-based frameworks. This technique analyzes the covariance of neural excitations on layers of interest and reduces the number of learnable parameters by modifying layers to operate in a projective space. Although the operations of the layer take place in a typically lower rank projective space, the expressivity of the layer remains high as the width—i.e., the number of neural activations—remains unchanged when compared to the original network architecture. This can be used in place of or in addition to pruning and quantization.

In this example, we'll take a look at the LSTM model used in the previous sections. In this case, there was **no significant loss in accuracy** after compressing the LSTM model and fine-tuning it with a few epochs of retraining. Compression with neural network projection reduced the LSTM model's memory footprint by 93% and reduced its inference time (latency) by about 40%.

Table of Contents

- Analyze LSTM Network for Compression
- Load input data
- Model Compression - Projection
 - Exploring Compression Levels
 - Compress LSTM Network 93% with Projection
 - Fine-tuning projected LSTM network
 - Compare accuracies of original and projected models
 - Compare inference speeds of original and projected models

Analyze LSTM Network for Compression

Let's load the LSTM network we've been using inside Deep Network Designer and analyze it for compression.

```
1 clear
2 load DLmodel_LSTM.mat
3 deepNetworkDesigner(lstm_Network)
```

DESIGNER

New Duplicate Copy Unlock Fit Zoom In Zoom Out Auto Analyze Analyze for Export

Editor: 90% UTF-8 LF Script

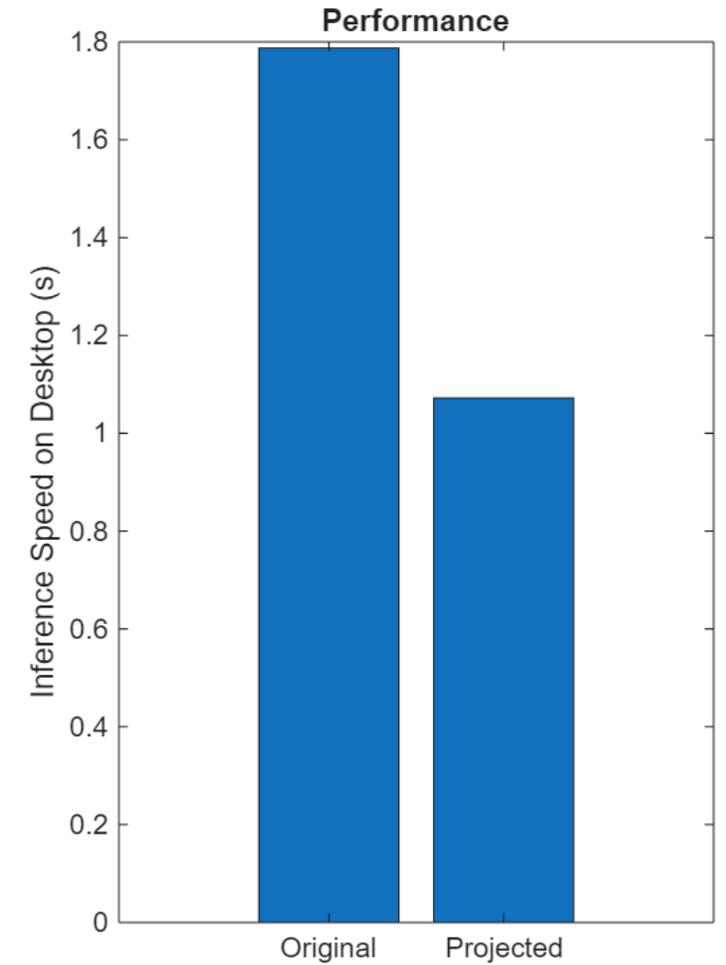
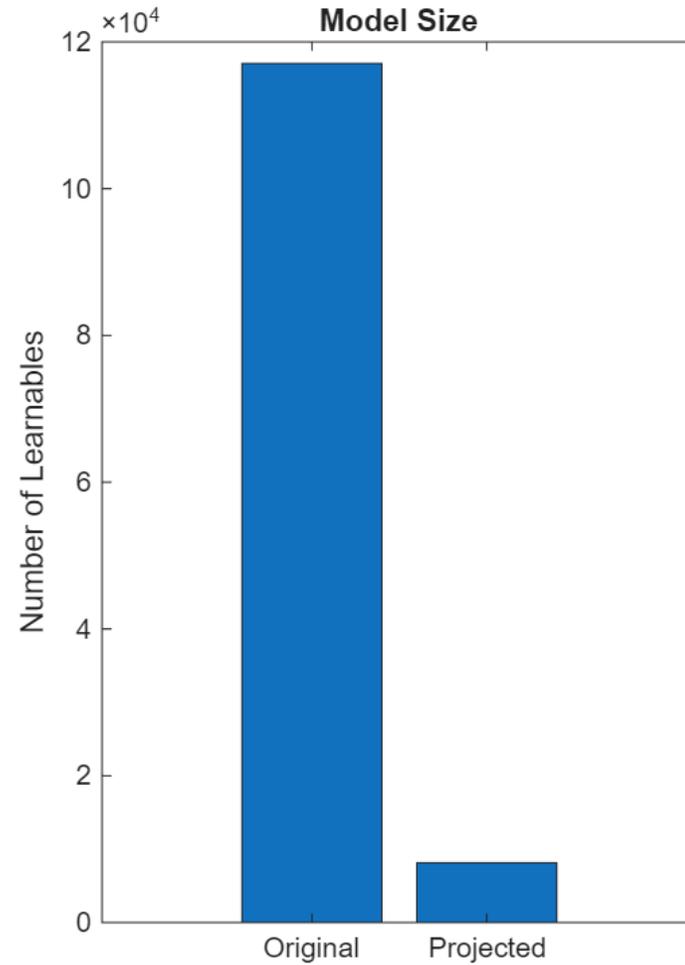
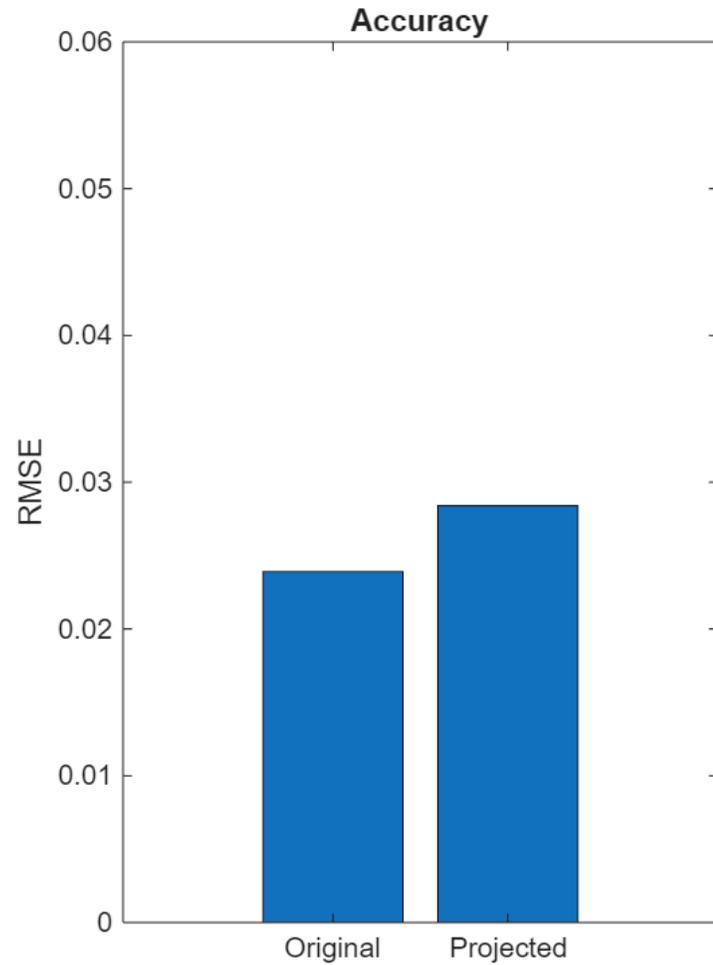
Data Preparation

AI Modeling

Simulation & Test

Deployment

Neural Network Projection Results



Manage AI tradeoffs for your system

	EKF Extended Kalman Filter	Tree Fine Regression Tree	FFN 1-hidden layer Feedforward Network	LSTM Stacked Long Short-Term Memory Network	LSTM* * Compressed Stacked Long Short-Term Memory Network
Preprocessing effort	●	●	●	●	●
Training Speed	N/A	●	●	●	●
Interpretability	●	●	●	●	●
Inference Speed	●	●	● ●	●	●
Model Size	●	●	●	●	●
Accuracy (RSME)	●	●	●	● ●	● ●

Results are specific to the Battery SoC Estimation Example



Agenda

- I. Introduction to AI with Model-Based Design
- II. Deep dive of Virtual Sensor Modeling workflow
- III. Resources for further learning

Topics we covered today



AI modeling with machine learning and deep learning methods



Python-MATLAB interoperability



Integrate AI model into system-level simulation and test



AI model compression

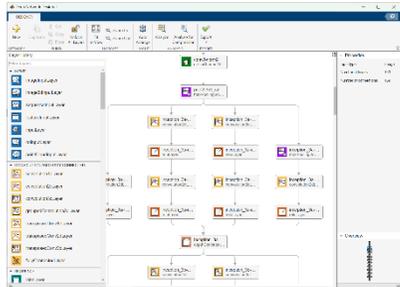
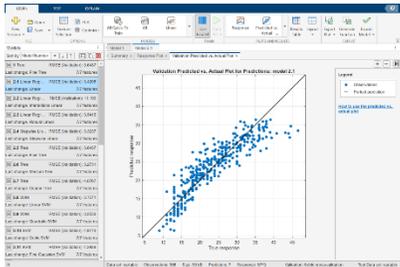


Deploy AI models to embedded target

Why MATLAB & Simulink?

- Low-code tools for AI modeling
- Easily integrate AI models into Model-Based Design
- Model Compression, Verification, and Cross-Platform Code generation
- Interoperability of Open-Source frameworks

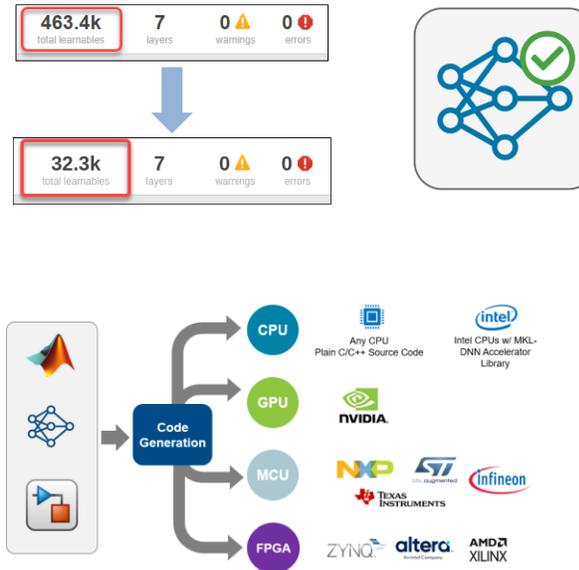
Low-code Apps



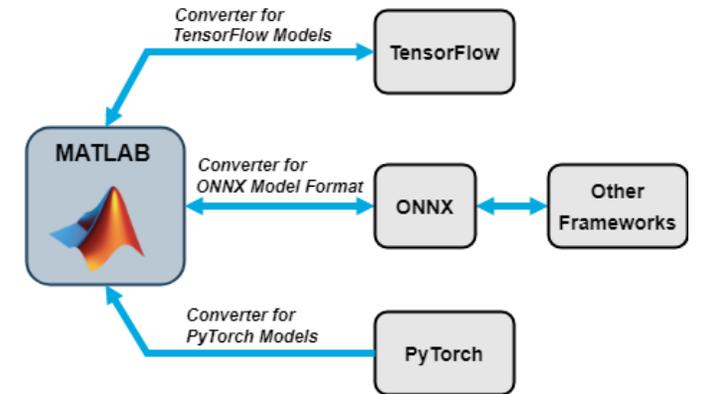
AI + Model-Based Design



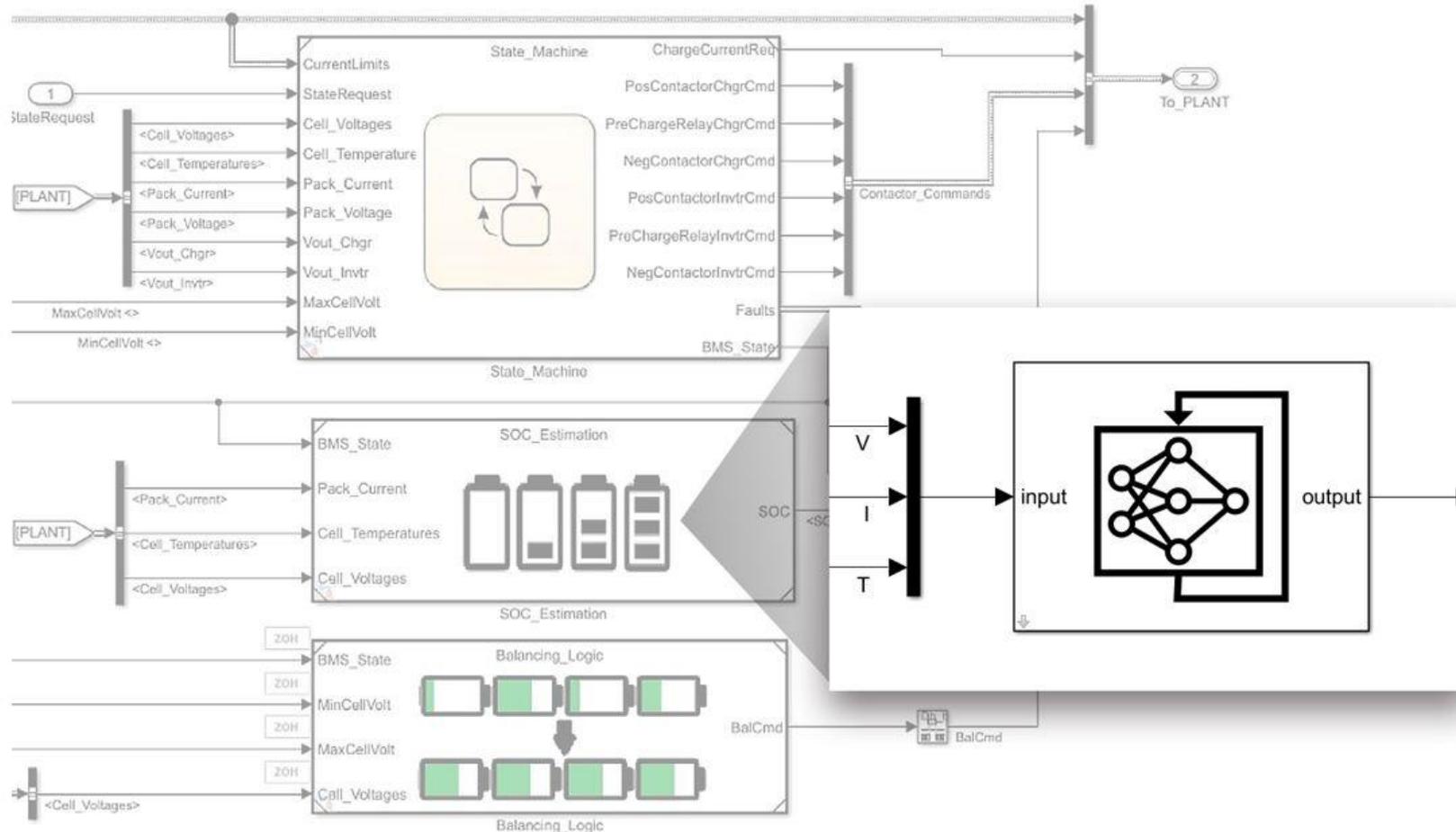
Compression, Verification, Code Generation



Interoperability



In summary, build a virtual sensor using AI and integrate into Simulink for system-level simulation and code generation



Topics we did not cover today



Model Comparison and Tuning



Explainable AI



AI Verification and Validation



And more... Let's follow up!

Resources for additional learning

- Other seminars and workshops
 - Reduced Order Modeling
 - Nonlinear system identification
 - Reinforcement Learning
 - Learning-based Controls
 - Hands-on virtual sensor modeling workshop
- AI with Model-Based Design
 - [Solutions Page](#)
- Self-paced trainings
 - Introduction:
 - [Simulink onramp](#)
 - [Stateflow onramp](#)
 - [Simscape onramp](#)
 - [Machine learning onramp](#)
 - [Deep learning onramp](#)
 - [Reinforcement learning onramp](#)
 - In-depth trainings:
 - [Simulink](#)
 - [Machine learning](#)
 - [Deep learning](#)
- Instructor-led trainings
 - [AI, Data Science, and Statistics](#)