AI Implementation based on compiling neural networks from SCADE language

December 2, 2021

Max Najork, Principal Engineer

Supported by:
Bernard Dion, Ansys Fellow
Jean-Louis Colaço, Distinguished Engineer

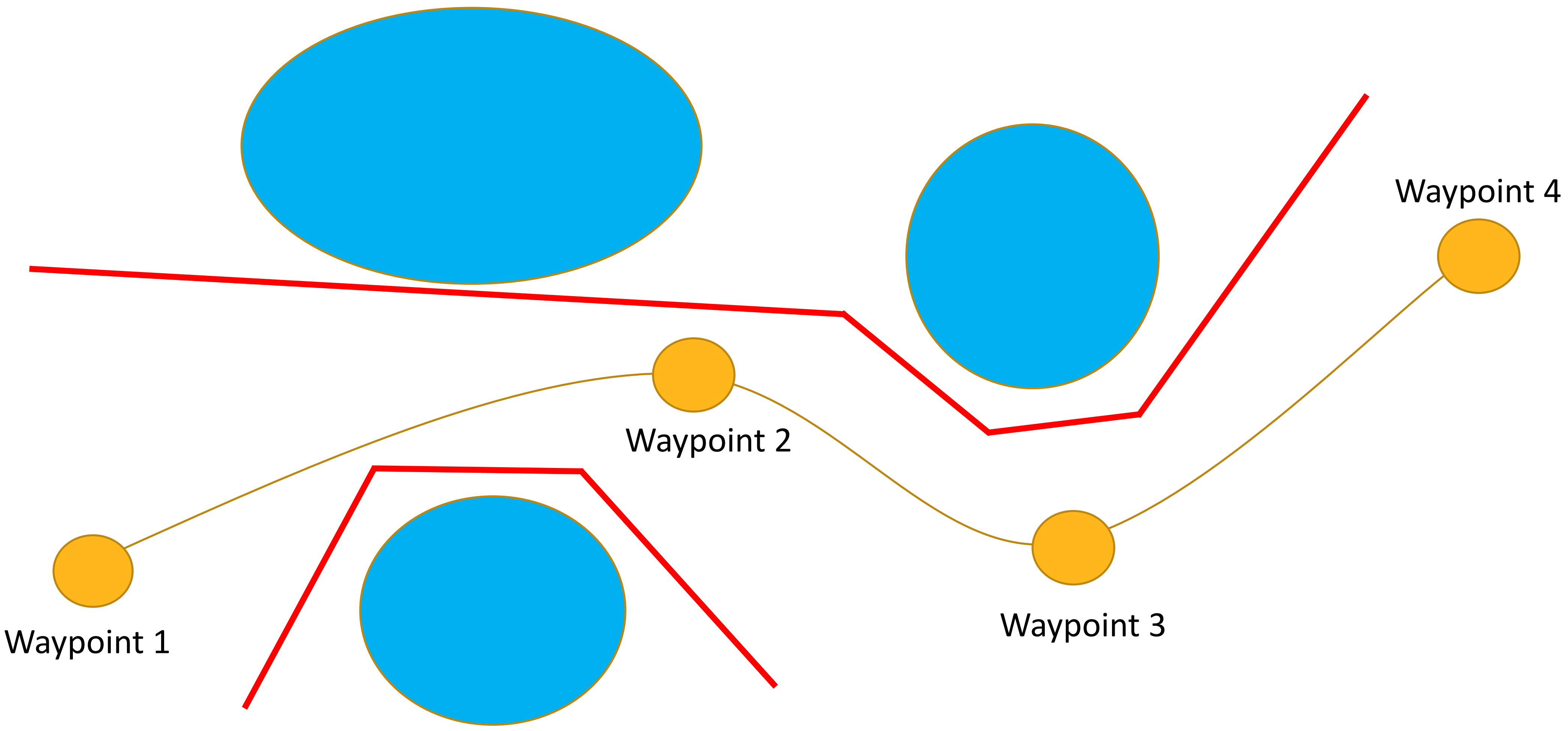
Agenda

1. Proposed AI workflow
2. Considerations on NN representation
3. Overview of the SCADE language and its code generation capabilities
4. SCADE-based neural network implementation flow
5. Neural network certification aspects
6. Summary & Conclusion

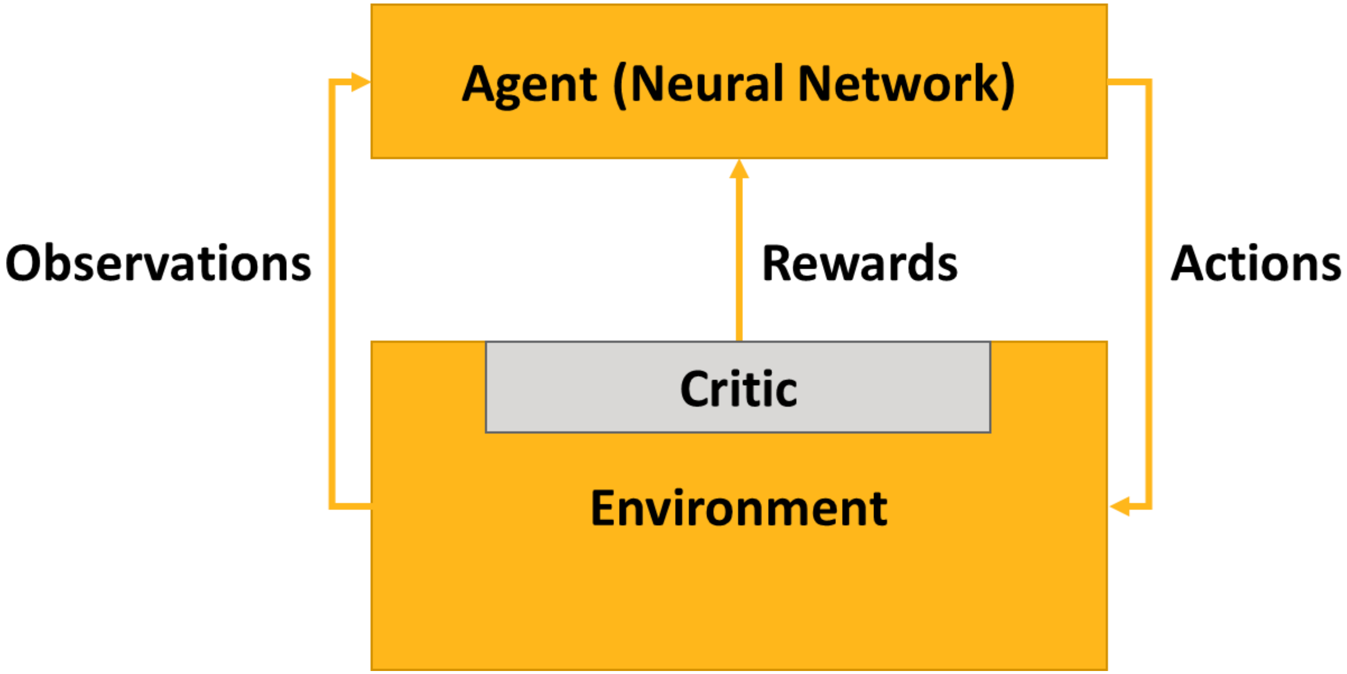
Proposed AI workflow

Customers Face New Challenges in Guidance, Navigation, and Control (GNC)

Guidance, Navigation, and Control (GNC)



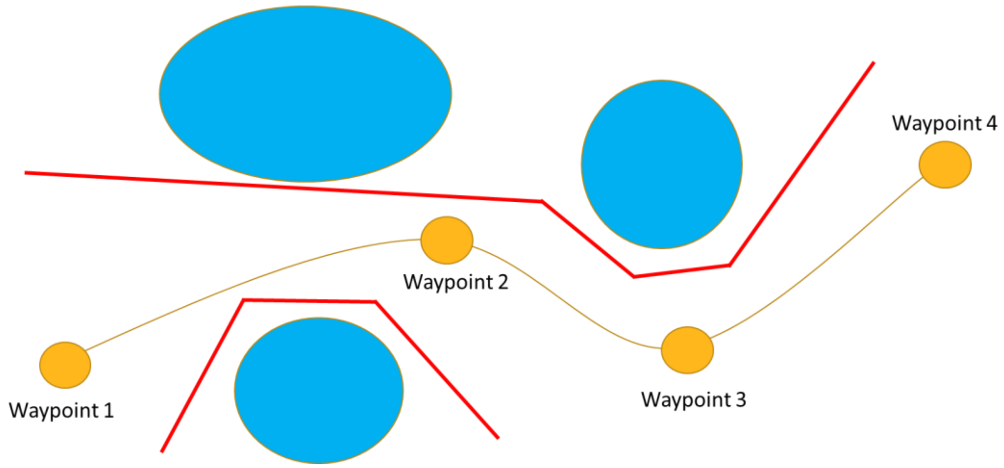
Deep Reinforcement Learning



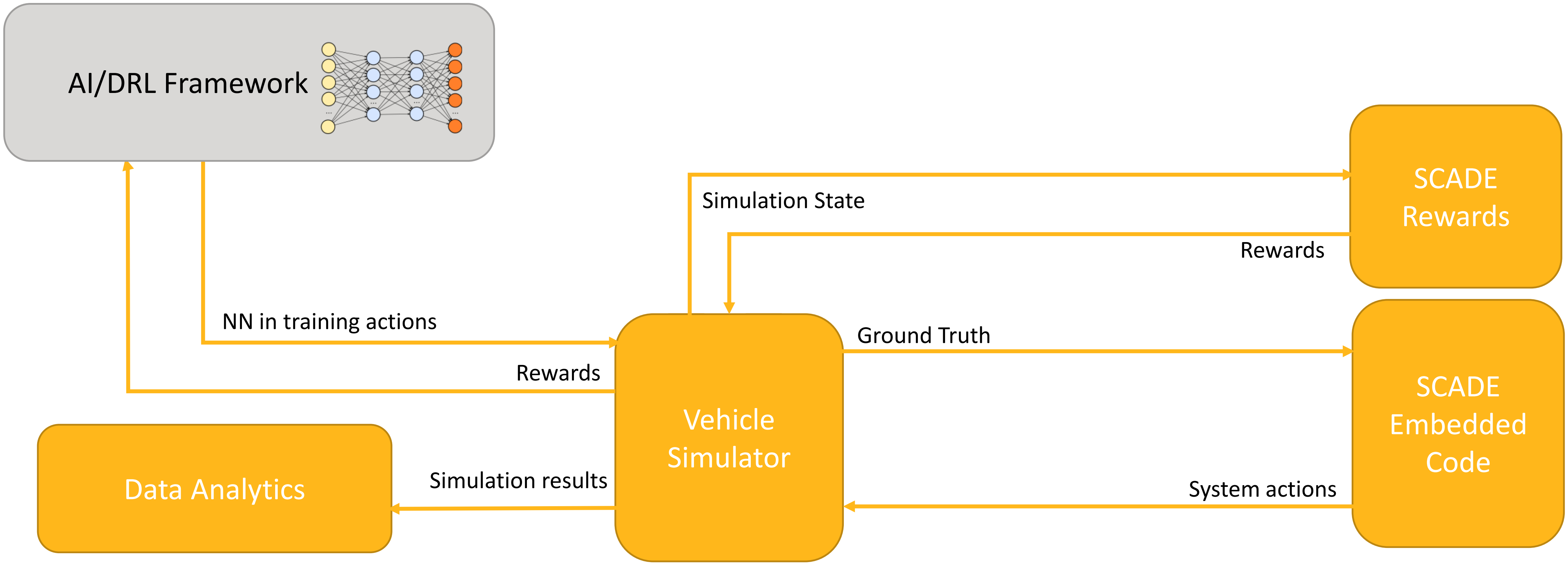
Deep Reinforcement Learning for GNC

- Observations:
  - Own Position and Rotation
  - Direction and distance to next waypoint
- Actions:
  - Desired Thrust
  - Desired Roll, Pitch, Yaw
- Rewards:
  - Positive reward whenever waypoint is reached
  - Highly negative when an obstacle is hit
  - Slightly positive when the distance to the waypoint is reduced

→ We can come up with any sort of reward



Training the Neural Network (Deep Reinforcement Learning)



Training the GNC Neural Network (1st Round)

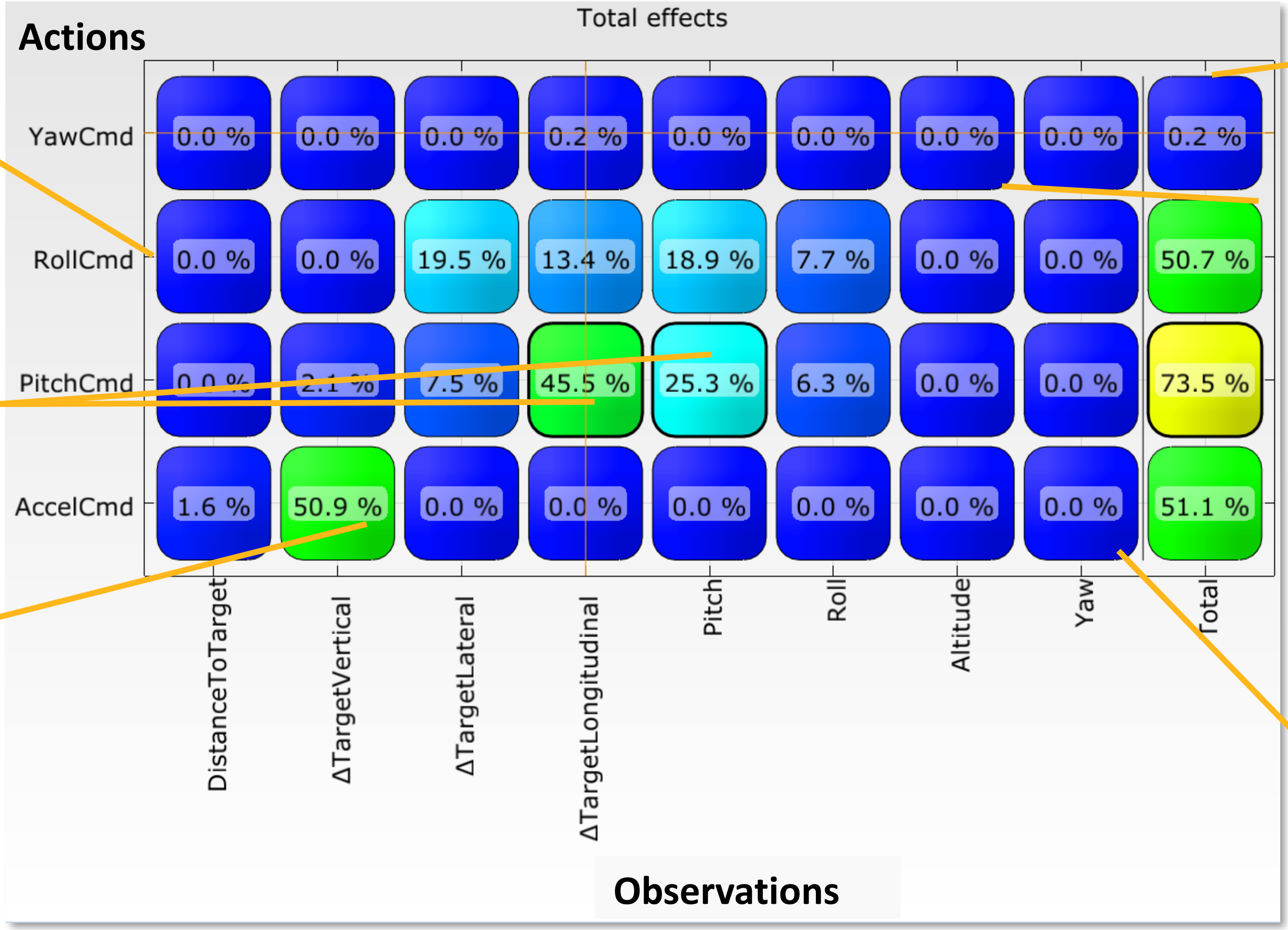
Video shows only snap shots from the training phase

Learning Issue: Roll and Yaw Command not learned properly

Roll command was **not properly learned**

Pitch command **was learned** based on **longitudinal distance** and **current pitch** angle

Acceleration command was learned based on the **difference in altitude** to the target



Yaw command was **not learned**

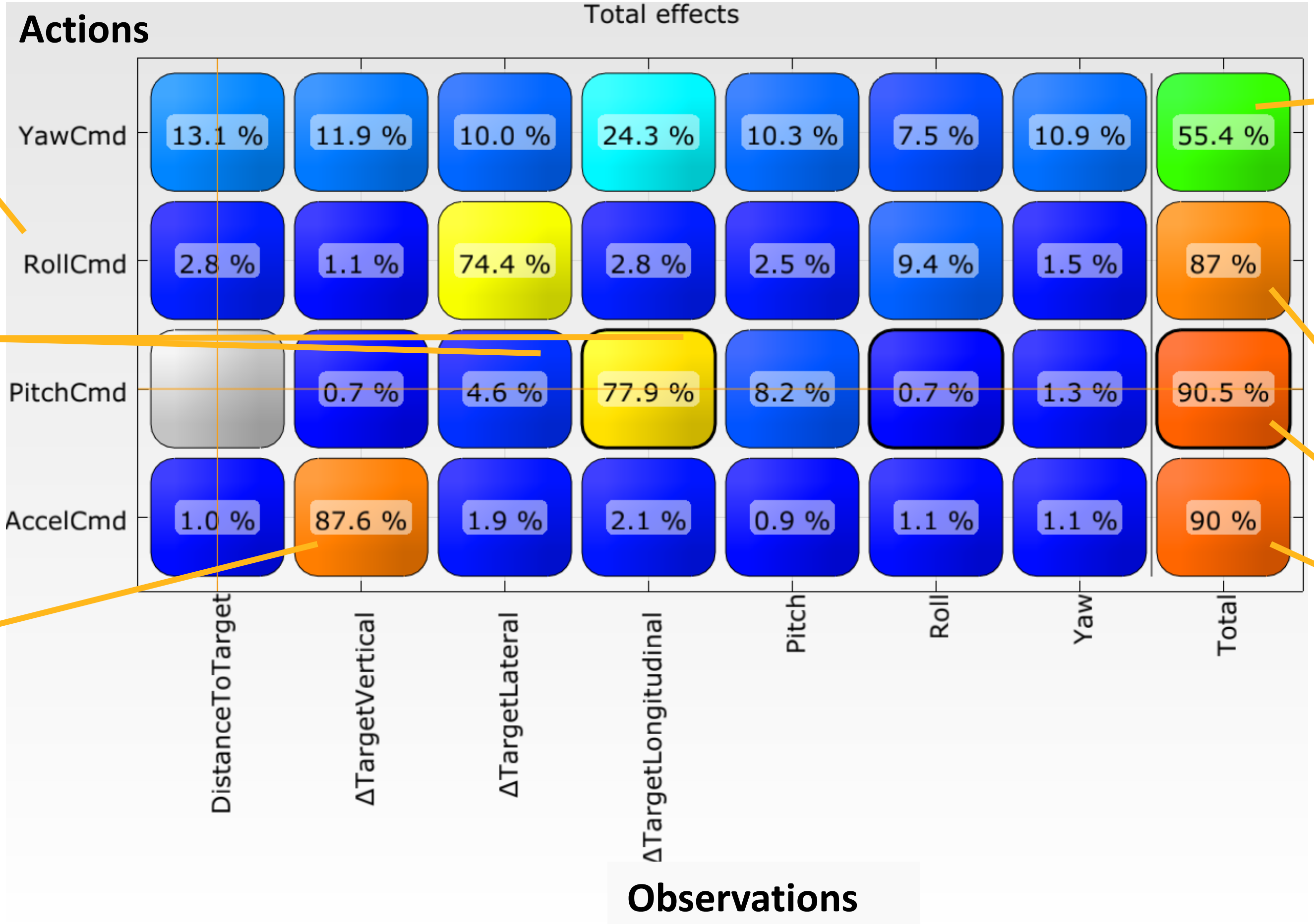
Measured yaw angle is **not used** by neural network

Training the GNC Neural Network (2nd Round)

Learning Issue: fixed (Roll and Yaw Command learned properly)

Roll command was **was learned** based on

Pitch command **was learned** based on **longitudinal distance** and **current pitch** angle

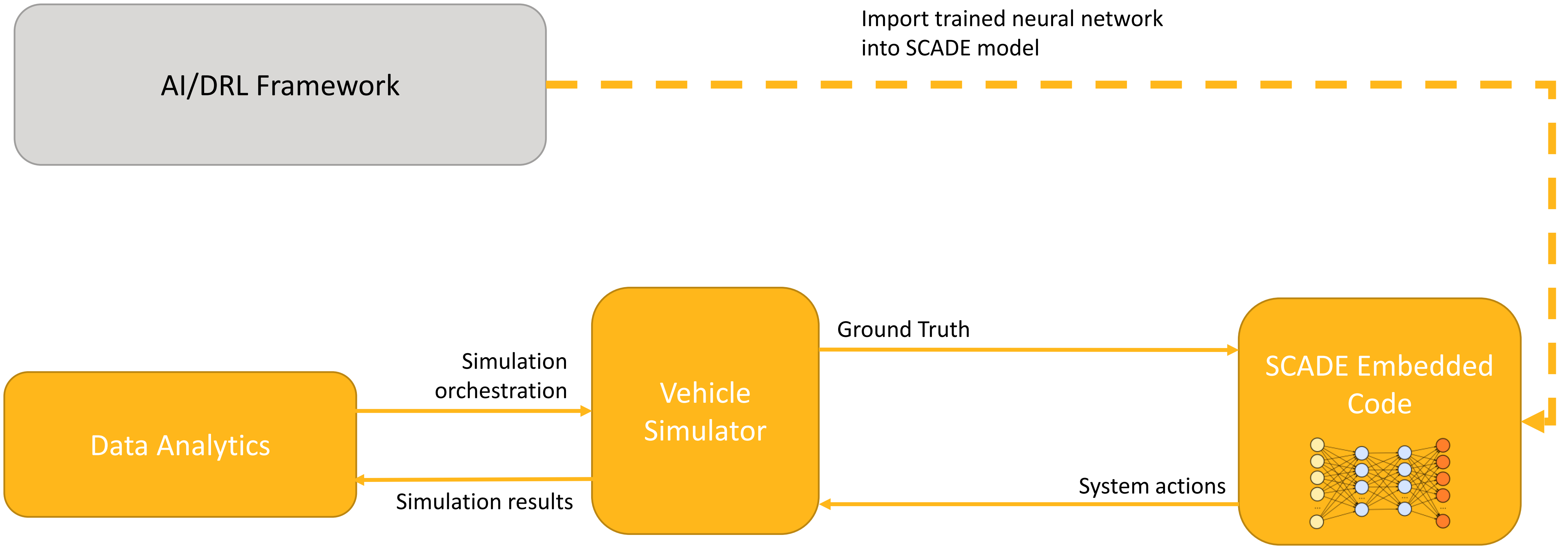


Yaw command **improved**

Acceleration command was learned based on the **difference in altitude** to the target

Higher Explainability of commands

Importing the NN into SCADE for Integration and Validation

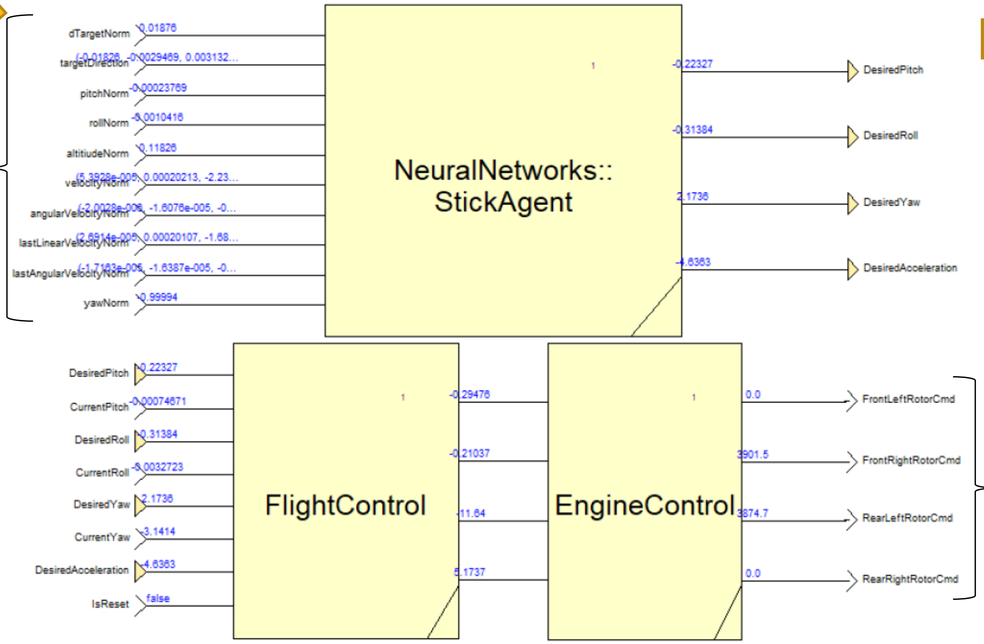


Validation of the Vehicle Function

Aircraft sensors deliver software inputs

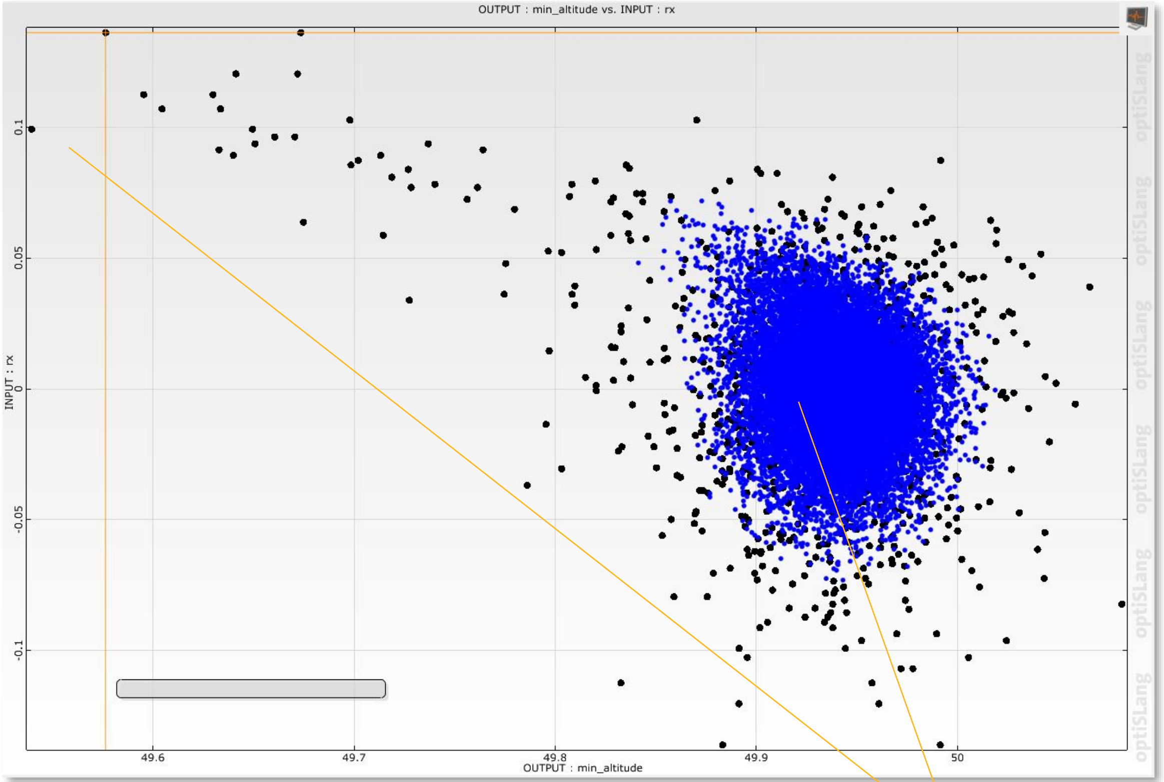
Aircraft actuators receive software outputs

Measurements of pitch, roll, yaw, etc.



Commands for aircraft movement and acceleration

Reliability Analysis: Zero failures, large separating distance



Method : Adaptive Response Surface Method (ARSM)

Complete approximations : 3 / 3
Selected data : 3. Approximation

Probability of Failure : 0
Standard deviation error : inf
Reliability Index : 10

Approximation errors : R^2 / R^2pred
lim_st_min_altitude: 1 / 0.809582

Number of designs
Total : 1000
Safe domain : 1000
Unsafe domain : 0
Failure strings : 0
Failed : 0

Directional Sampling on Response Surface

Complete directions : 5000 / 5000

Number of designs
Total : 25001
Safe domain : 25001
Unsafe domain : 0
Failure strings : 0
Failed : 0

Not a single scenario was unsafe!

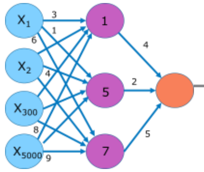
Safety Limit 20m

Minimum altitude for each scenario

No scenario was lower than 49 meters!

Neural Network Import to SCADE and Safe Software Generation

Trained model



TensorFlow

PYTORCH

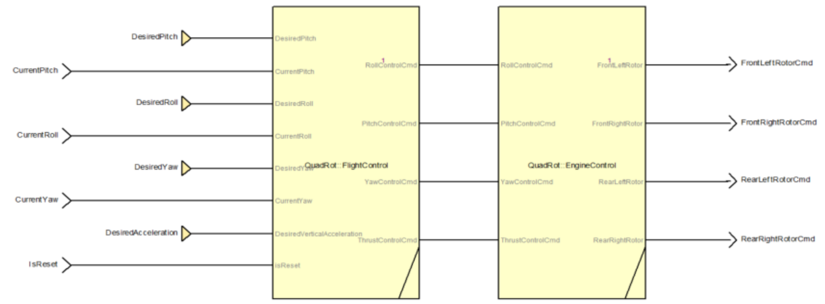
Caffe

Importer

Neural Network in SCADE

```
function #pragma kcg separate_io #end model(observations : float32^8) returns(fc_out : float32^8)
var
  fc_2 : float32^256;
  fc_1 : float32^256;
let
  fc_out = (Layers::Dense <<256,8>>)(fc_2, fc_out_kernel, fc_out_bias);
  fc_2 = (Layers::TanH <<256>>)((Layers::Dense <<256,256>>)(fc_1, fc_2_kernel, fc_2_bias));
  fc_1 = (Layers::TanH <<256>>)((Layers::Dense <<8,256>>)(observations, fc_1_kernel, fc_1_bias));
tel

const
  imported fc_out_kernel : float32^8^256;
  imported fc_out_bias : float32^8;
  imported fc_2_kernel : float32^256^256;
  imported fc_2_bias : float32^256;
  imported fc_1_kernel : float32^256^8;
  imported fc_1_bias : float32^256;
```

Conventional Functions in SCADE



Generate Code & Compile

Considerations on NN representation

Transition from Neural Network Frameworks to Design Models

TensorFlow compute graph as Protobuf containing full learning problem:

Formal representation of the same neural network in SCADE Language after import

```
function #pragma kcg separate_io #end policy_mlp(input : float32^8) returns(pi : float32^4)
var
  shared_fc0 : float32^64;
  shared_fc1 : float32^64;
let
  shared_fc0 = (Layers::TanH <<64>>)((Layers::Dense <<8,64>>)(input, shared_fc0_weight, shared_fc0_bias));
  shared_fc1 = (Layers::TanH <<64>>)((Layers::Dense <<64,64>>)(shared_fc0, shared_fc1_weight, shared_fc1_bias));
  pi = (Layers::Softmax <<4>>)((Layers::Dense <<64,4>>)(shared_fc1, pi_weight, pi_bias));
tel
```

Consistency of Models between Phases

Train → Validate → Implement

Consistency of models between the different phases is key to the safe operation of ML-based vehicle functions

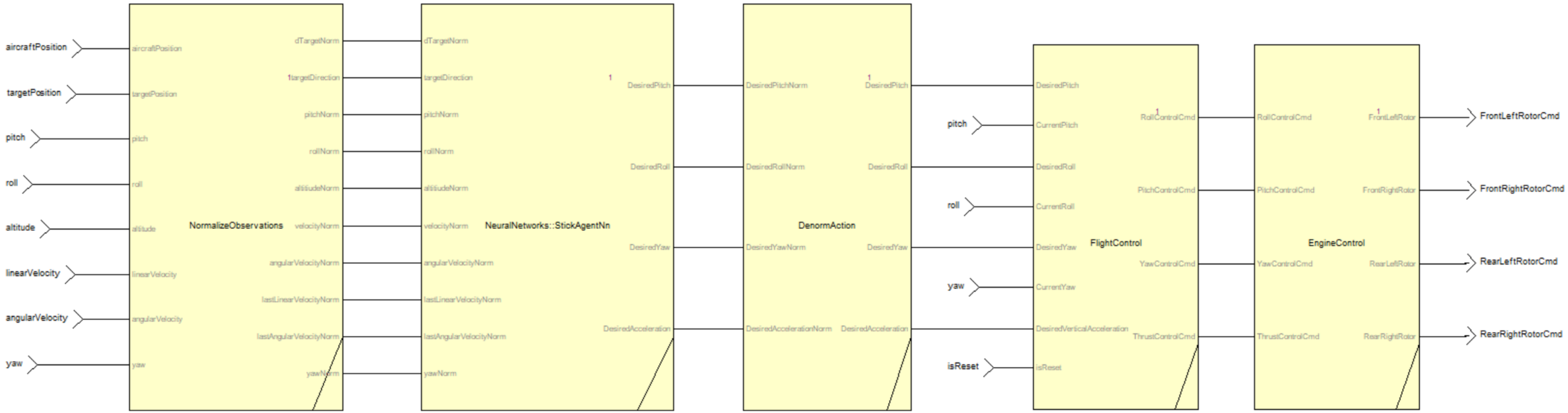
Embodiment through software

Observation normalization (traditional SW)

Neural Network (ML component)

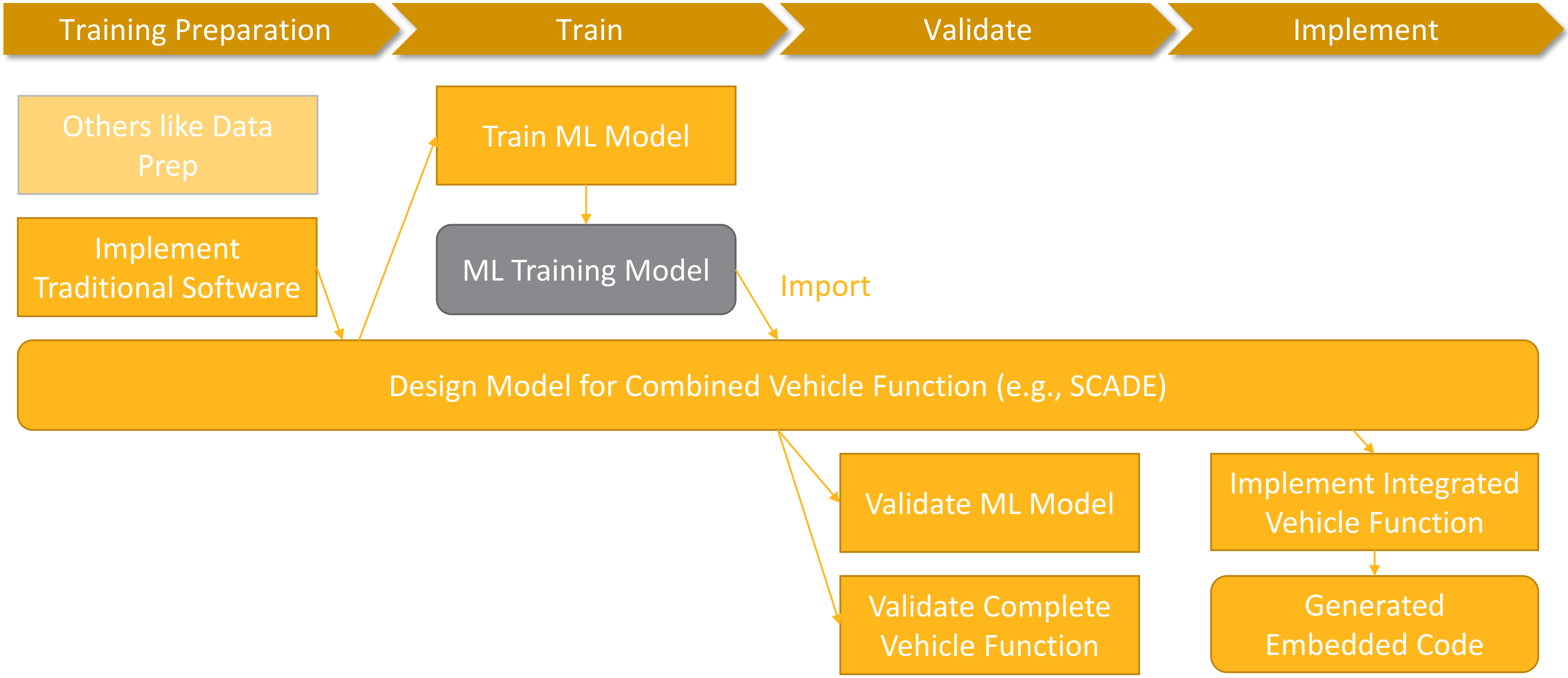
Action de-normalization (traditional SW)

Control laws (tradition software)



Consistency between training, validation, and implementation is ensured through integration with actual embedded models and qualified code generation

Using Design Models Across the Phases



SCADE Language and Code Generation Properties

Ansys

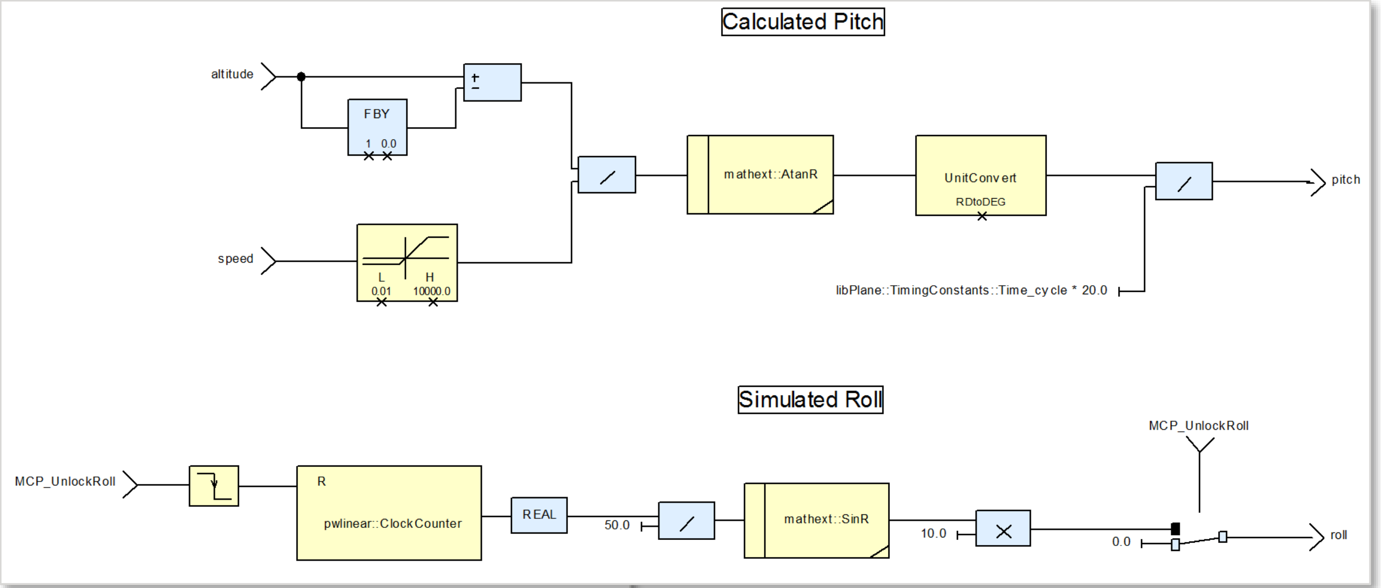
Scade

- **SCADE**: **S**afety **C**ritical **A**pplication **D**evelopment **E**nvironment.
- Domain specific language:
  - dedicated to real-time embedded software,
  - based on synchronous languages principles => parallel composition is deterministic,
  - defined and documented independently of toolset implementation,
  - focuses on safety, has strong statically guaranteed properties:
    - typed, safe arrays operations,
    - bounded in time and memory (no dynamic memory allocation),
    - defined output values (cannot depend on uninitialized memory),
    - parallelism schedulable as a static sequence.
- SCADE code generator (KCG) is qualified for DO-330 TQL-1

P. Caspi, N. Halbwachs, D. Pilaud, and J. Plaice. Lustre: a declarative language for programming synchronous systems. In *14th ACM Symposium on Principles of Programming Languages.* 1987.

J-L. Colaco, B. Pagano, and M. Pouzet. Scade 6: A Formal Language for Embedded Critical Software Development. In *Eleventh International Symposium on Theoretical Aspect of Software Engineering (TASE).* 2017.

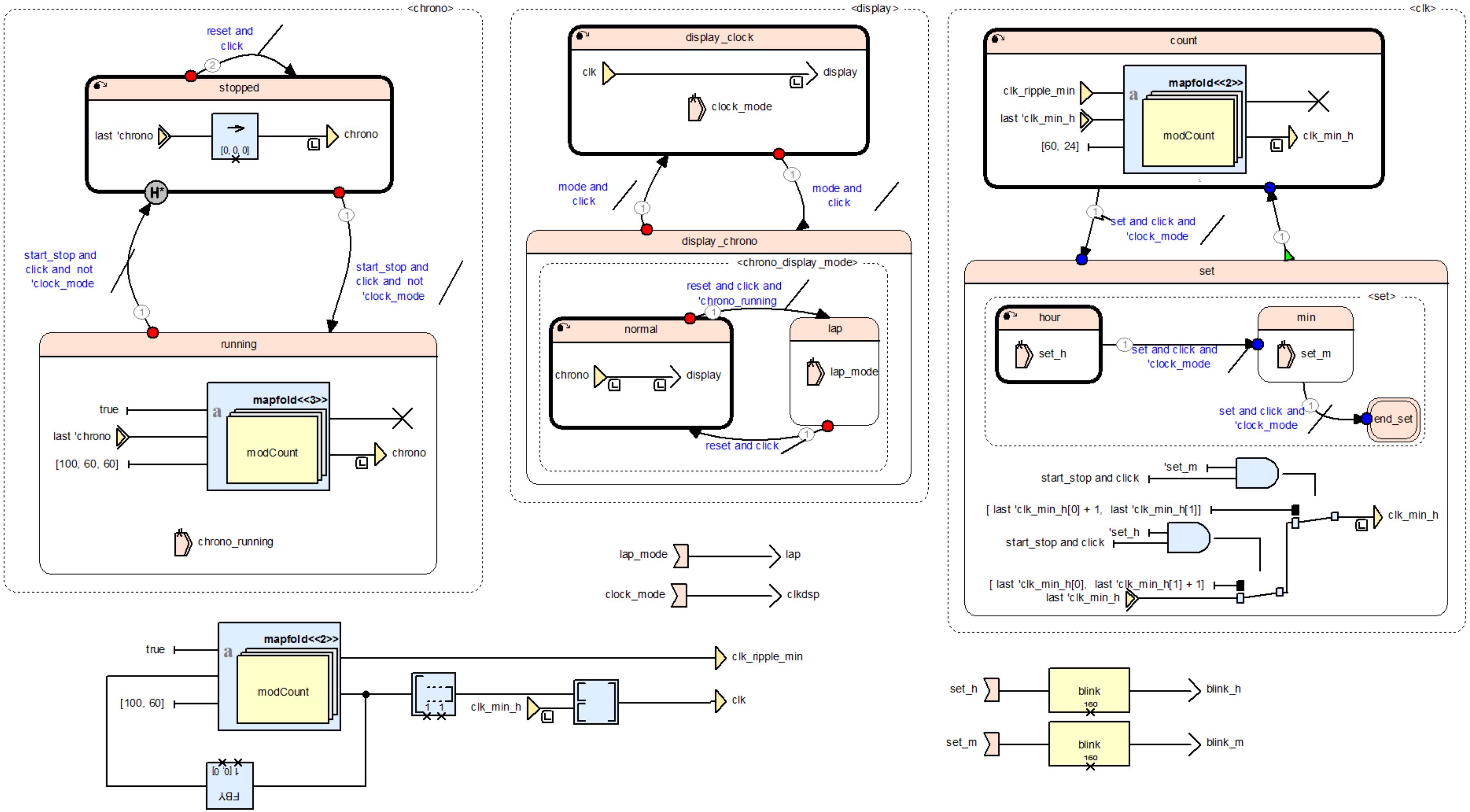
Available notations for SCADE Models




```
node ComputePitchRoll
  (speed : float32;
   MCP_UnlockRoll : bool;
   altitude : float32)
  returns (pitch, roll : float32)
let
    roll =
      if MCP_UnlockRoll
      then mathext::SinR(
             (pwlinear::ClockCounter(
                digital::FallingEdge(MCP_UnlockRoll)) : float32) / 50.0) * 10.0
      else 0.0;
    pitch =
      UnitConvert(mathext::AtanR(
        (altitude - (0.0 -> pre altitude)) /
        pwlinear::LimiterUnSymmetrical(speed, 0.01, 10000.0)),
        RDtoDEG) /
      (libPlane::TimingConstants::Time_cycle * 20.0);
tel
```

Both textual and graphical representation define the same operator

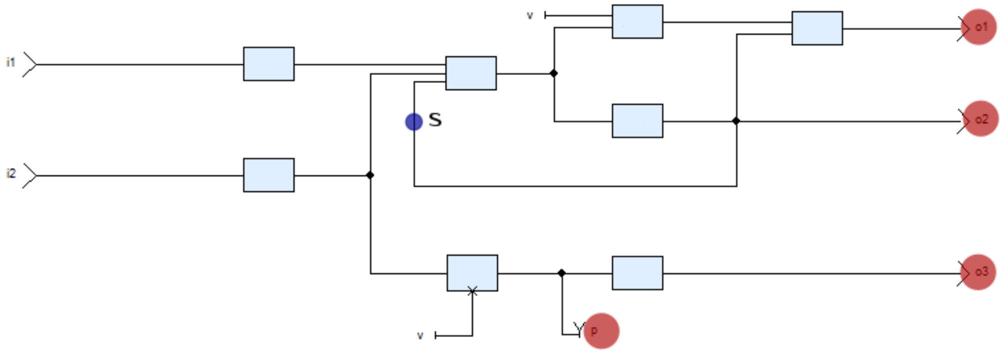
State machines and block diagrams in a single language



SCADE Suite KCG: a DO-330 TQL-1 qualified code generator

- Translates a SCADE model to C or Ada code.
- Qualification process
  - Aims at ensuring that the generated **code implements** the **function defined by the model**.
  - The generated code is **verified** wrt the semantics defined in the **language specification**.
  - Code **generation options** only apply to the shape of the generated code and **do not affect the function** specified by the model.
- Qualification credits allow to use this code without having to verify that it implements the function defined by the model.

Model Coverage for Scade design



- Based on a measure defined at model level: s (in blue) **is covered by a test** showing its **ability to contribute to one of the outputs** (in red).
- Generalizes the idea of masking MC/DC.
- A **100% coverage** analysis of the **model holds on the code** generated by KCG with the same test suite (DO-330 FAQ#11).
- TQL-4 tool that gives credit on both activities: code and model coverage analysis.
- Good for conventional functions, likely to be required for NN code but does not fully tackle the verification of the absence of unintended function.

SCADE Neural Network Implementation Flow

Ansys

Arrays in Scade and NN inference implementation

- Arrays main features:
  - Single dimension, nesting is allowed (arrays of arrays).
  - Safe:
    - size is part of the type and of the type checking;
    - accesses are always done within bounds (dynamic projections have a default).
    - arrays are always completely defined.
    - manipulated through iterators (map, fold, …)
- Polymorphism of user defined operators in types and array sizes.
- Expressive enough to specify standard NN layers.

Scade allows to:

- write generic libraries of NN layers,
- compose them to define NN-based function and
- take certification credits of the tools (see SCADE DO-178C handbook).

Scade library of NN layers

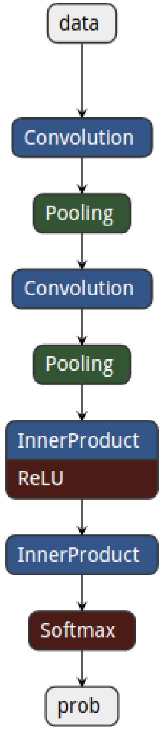
Defined with the textual notation, more convenient here:

```
-- InnerProduct layers

function InnerProduct_3D << D3, D2, D1, D_o >> (x : 'T^D1^D2^D3; weight : 'T^D1^D2^D3^D_o; bias : 'T^D_o)
returns (y : 'T^D_o) where 'T numeric
  y = (map (fold (fold (_dot_bias <<D1>>) <<D2>>) <<D3>>) <<D_o>>)(bias, x^D_o, weight);

function InnerProduct_1D << N, M >> (x : 'T^N; weight : 'T^N^M; bias : 'T^M)
returns (y : 'T^M) where 'T numeric
  y = (map (_dot_bias <<N>>) <<M>>)(bias, x^M, weight);


-- ReLU activation layer

function relu(x : 'T) returns (y : 'T) where 'T numeric
  y = if x >= 0 then x else 0;

function ReLu << N >> (x : 'T^N) returns (y : 'T^N) where 'T numeric
  y = (map relu <<N>>)(x);


-- Softmax layer, the normalized exponential function

function Softmax << N >> (x : 'T^N) returns (y : 'T^N) where 'T float
var m, sum : 'T;
    n, E   : 'T^N;
let
    m = (fold max <<N>>)(x[0], x);
    n = (map  $-$ <<N>>)(x, m^N);
    E = (map  exp <<N>>)(n);
  sum = (fold $+$ <<N>>)(0., E);
    y = (map  $*$ <<N>>)(E, (1. / sum)^N);
tel
```

Example: LeNet-5

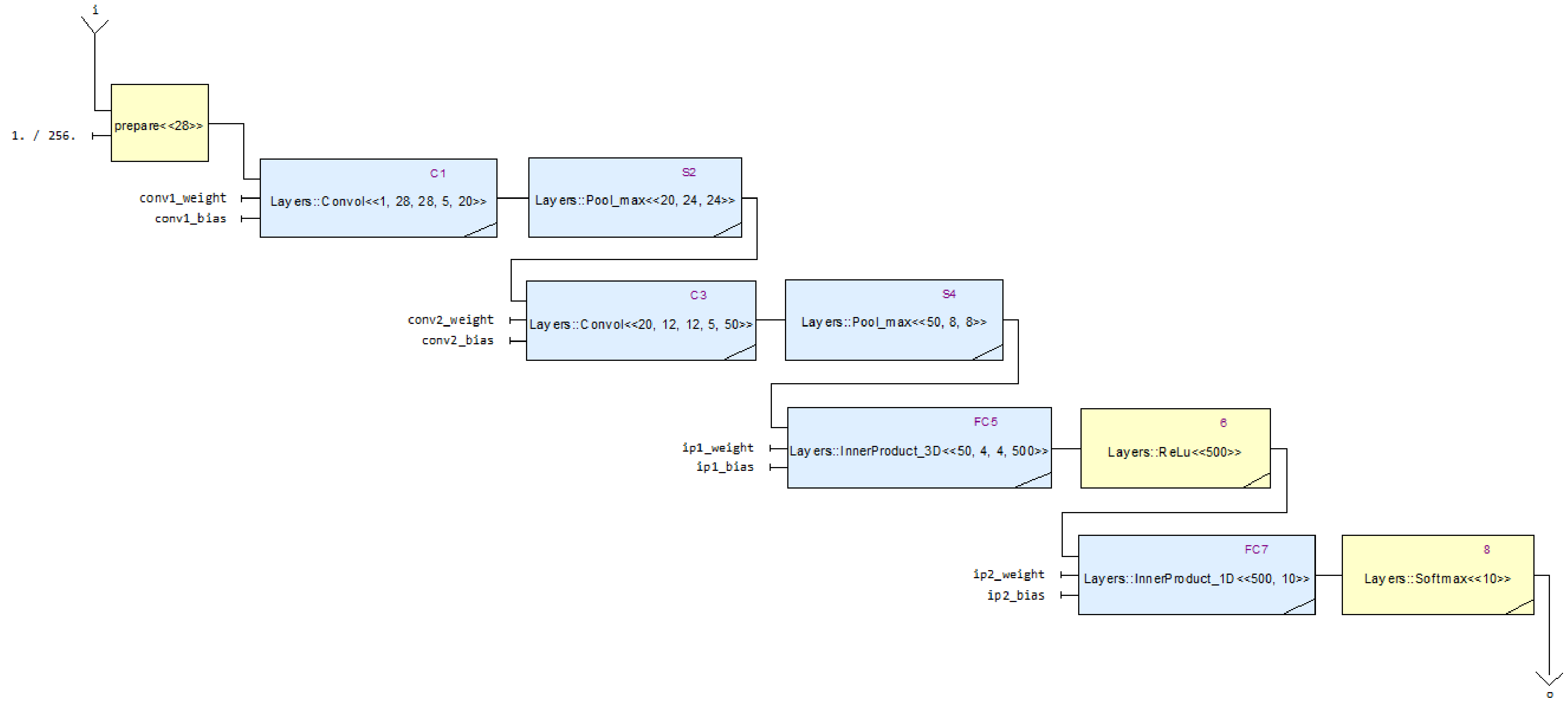
```
function lenet(i : uint8^(28*28)) returns (o : float32^10)
var L0      : float32^28^28^1;
    L1      : float32^24^24^20;
    L2      : float32^12^12^20;
    L3      : float32^8^8^50;
    L4      : float32^4^4^50;
    L5, L6 : float32^500;
    L7      : float32^10;
let
  o  = (Layers::Softmax              << 10 >>)(L7);
  L7 = (Layers::InnerProduct_1D     <<500, 10>>)(L6, ip2_weight, ip2_bias);
  L6 = (Layers::ReLu                 << 500 >>)(L5);
  L5 = (Layers::InnerProduct_3D <<50,4,4, 500>>)(L4, ip1_weight, ip1_bias);
  L4 = (Layers::Pool_max             <<50,8,8>>)(L3);
  L3 = (Layers::Convol       <<20,12,12, 5,50>>)(L2, conv2_weight, conv2_bias);
  L2 = (Layers::Pool_max           <<20,24,24>>)(L1);
  L1 = (Layers::Convol        <<1,28,28, 5,20>>)(L0, conv1_weight, conv1_bias);
  L0 = (prepare <<28>>)(i);
tel
```

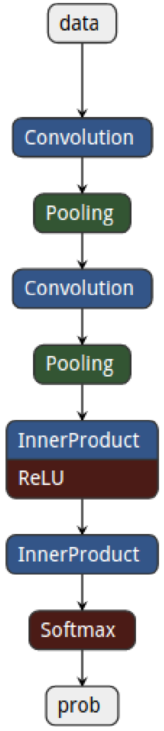
```
-- model parameters
const imported conv1_weight : float32^5^5^1^20;    --     500
      imported conv1_bias   : float32^20;          --      20
      imported conv2_weight : float32^5^5^20^50;   --   25000
      imported conv2_bias   : float32^50;          --      50
      imported ip1_weight   : float32^4^4^50^500;  -- 400000
      imported ip1_bias     : float32^500;         --     500
      imported ip2_weight   : float32^500^10;      --    5000
      imported ip2_bias     : float32^10;          --      10
-- in total: 431080 parameters
```



LeNet : a CNN for «*handwritten and machine-printed character recognition*».
by Y. LeCun, L. Bottou, Y. Bengio (1998)

Example: LeNet-5 (equivalent diagram)





Neural Network Certification Aspects

Ansys

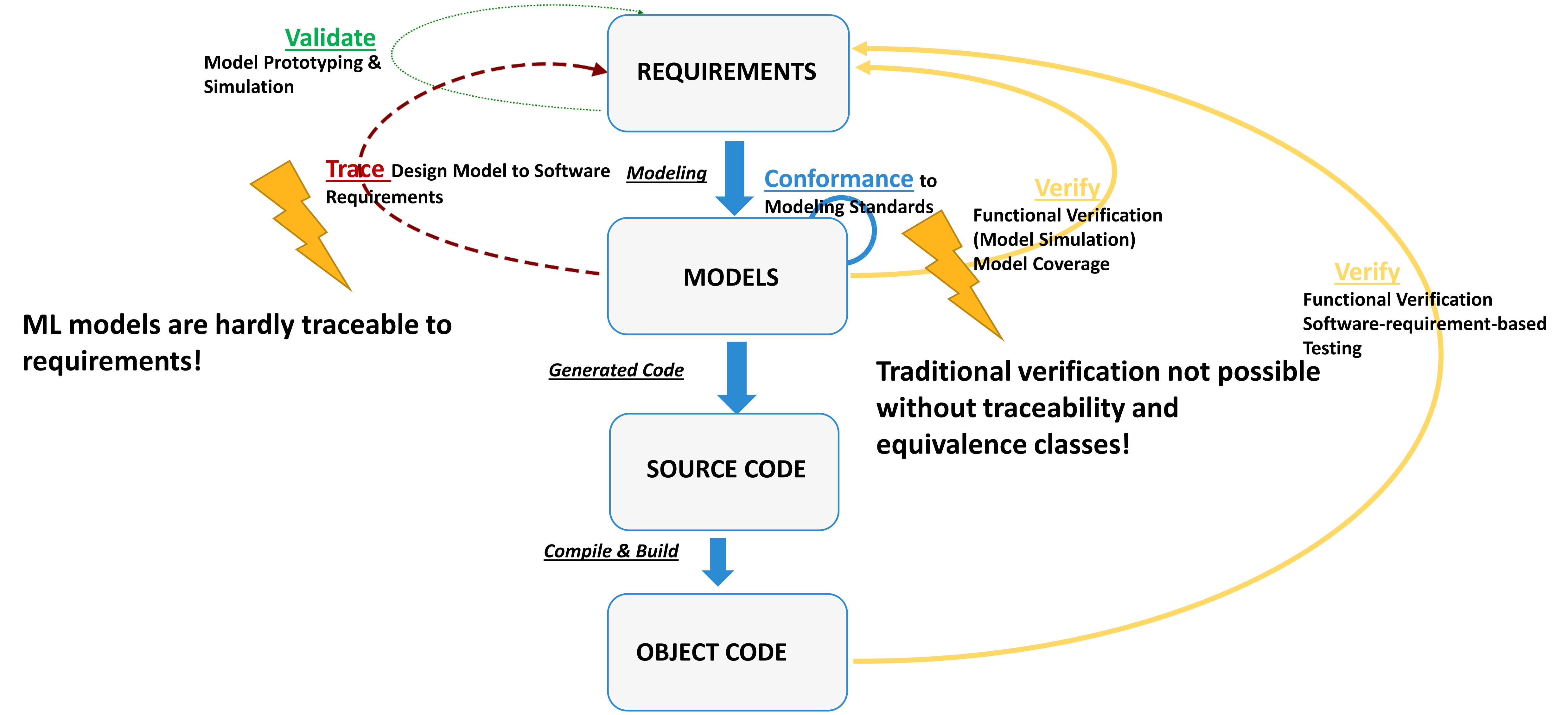
Approach: Positioning of the ML model within the DO-178C/ DO-331 life-cycle

From RTCA DO-331:

Table MB.1-1 Model Usage Examples

<table>
<tr><th>Process that generates the life-cycle data</th><th>MB Example 1</th><th>MB Example 2</th><th>MB Example 3</th><th>MB Example 4
(See Note 1)</th><th>MB Example 5
(See Note 1)</th></tr>
<tr><td rowspan="2">System Requirement and System Design Processes</td><td rowspan="2">Requirements allocated to software</td><td rowspan="2">Requirements from which the Model is developed</td><td rowspan="2">Requirements from which the Model is developed</td><td rowspan="2">Requirements from which the Model is developed</td><td>Requirements from which the Model is developed</td></tr>
<tr><td rowspan="3">Design Model</td></tr>
<tr><td rowspan="2">Software Requirement and Software Design Processes</td><td>Requirements from which the Model is developed</td><td>Specification Model
(See Note 2)</td><td>Specification Model</td><td rowspan="2">Design Model</td></tr>
<tr><td>Design Model</td><td>Design Model</td><td>Textual description
(See Note 3)</td></tr>
<tr><td>Software Coding Process</td><td>Source Code</td><td>Source Code</td><td>Source Code</td><td>Source Code</td><td>Source Code</td></tr>
</table>

DO-178C/DO-331 generic model-based workflow (for ML)



Ansys

Model Simulation during Validation Activities

Formal Model for Combined Vehicle Function (e.g., SCADE)

Code is guaranteed (tool qualification) to comply with functionality specified in model

Generated Traditional Host Testing Code

Generated Neural Network Code

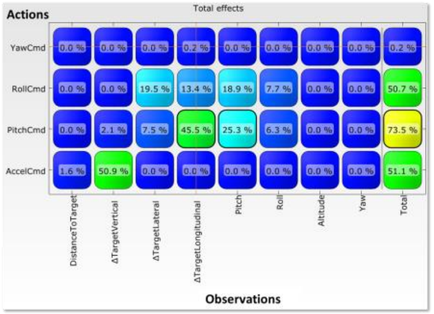
Generated Vehicle Function Code

Generated Target Code

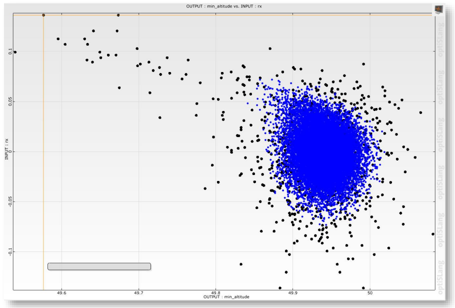
Traditional Host-based Testing

Step	Name	Actual Value	Expected Value
1	AutoPilot/DesiredVerticalAcceleration	0.0	0.0
1	AutoPilot/DesiredRoll	0.002504999982	2.505e-3
1	AutoPilot/DesiredPitch	0.005004999693	5.005e-3
2	AutoPilot/DesiredVerticalAcceleration	580.0	580.0
2	AutoPilot/DesiredRoll	-1.697490096	-1.697
2	AutoPilot/DesiredPitch	-1.19498992	-1.195
3	AutoPilot/DesiredVerticalAcceleration	80.0	80.0
3	AutoPilot/DesiredRoll	-1.697485089	-1.697
3	AutoPilot/DesiredPitch	-1.194984913	-1.195

Validate ML Model



Validate Complete Vehicle Function



Software Behavior and Numerical Computations

- Model Simulation is largely accepted to demonstrate the compliance of a model with its requirements
- However, RTCA DO-331 requires certain aspects to be tested on target (e.g., numerical accuracy)

→NN may be strongly sensitive towards exactly these differences

→NN numerical robustness is a key requirement to proof complementing model simulation

Summary and Conclusion

Summary and Conclusion

- AI-based vehicle functions allow us to **increase the level of autonomy**
- AI **certification remains challenging** but is progressing quickly
- We propose the following **flow for verification and safe implementation**:

Trained NN Model → Obtain Formal Software Model → Understand Learning Result → Quantify Failure Probability → Utilize Qualified Code Generation → Safe Source Code

Ansys

Max.Najork@ansys.com
Jaehoon.lim@ansys.com