UNIVERSITÀ
DI TRENTO

Technical
University
of Munich

NNodely: An Open Framework for Model-Structured Neural Networks in Robotics

September 29^{th}, 2025

Prof. Gastone Pietro Rosati Papini
Dr. Mattia Piccinini

University of Trento & Technical University of Munich

Overview

1. Motivation

1.1 Beyond Black-Box Neural Networks

1.2 Physics-Guided Learning

2. NNodely

2.1 Model-Structured Neural Networks (MS-NNs)

2.2 Framework

3. Use Cases

4. Conclusions

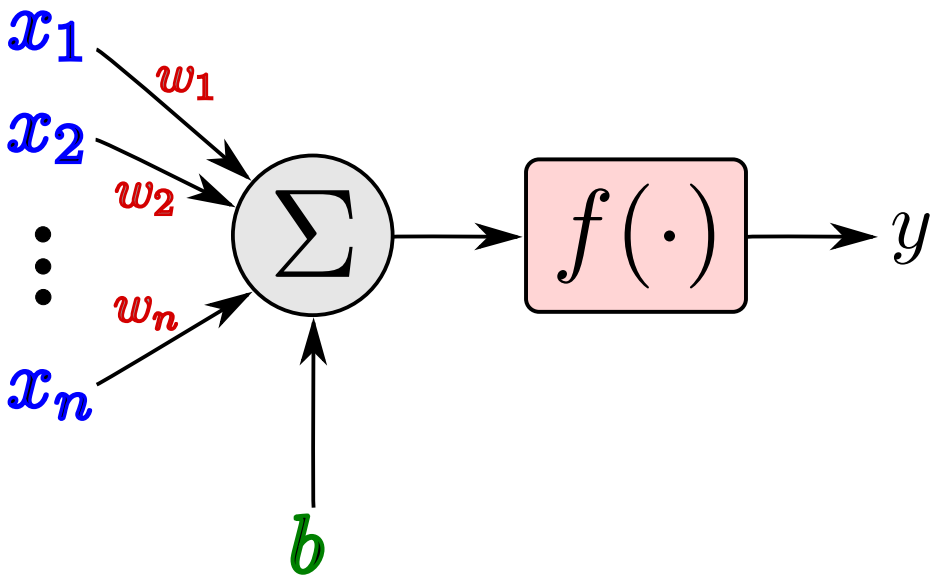
Motivation

Motivation

Beyond Black-Box Neural Networks

The Perceptron

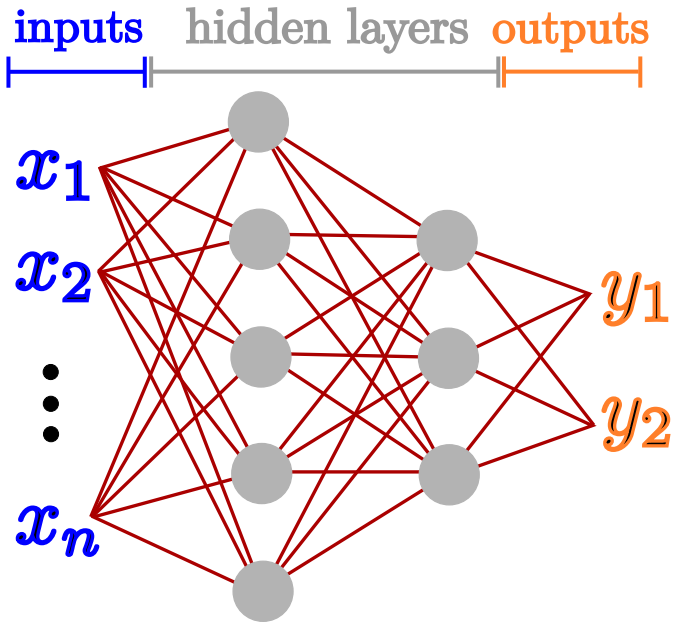
The basic component of a **neural network** (NN) is the **Perceptron**[1]



$$y = f\left(\sum_{k=1}^{n} x_k \cdot w_k + b\right)$$

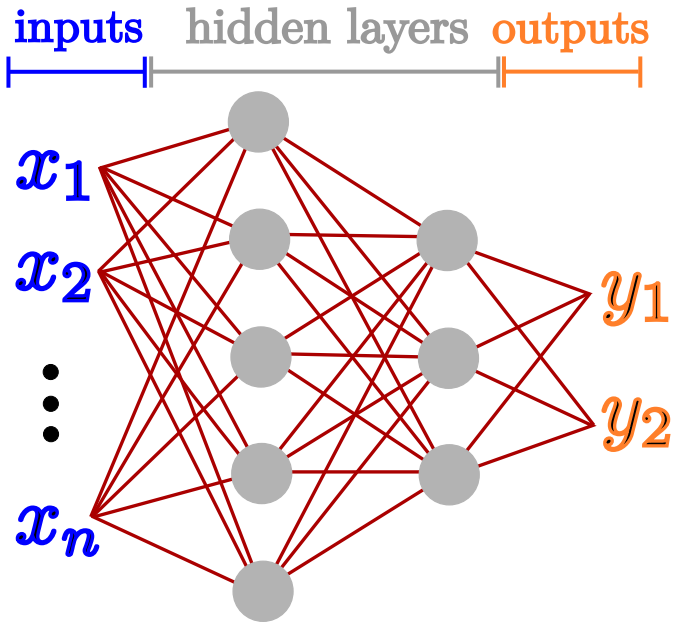
[1]Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain.". In: *Psychological review* (1958).

Multi-layer Perceptron (MLP)



[2]George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Math. Control Signal Systems* (1989).

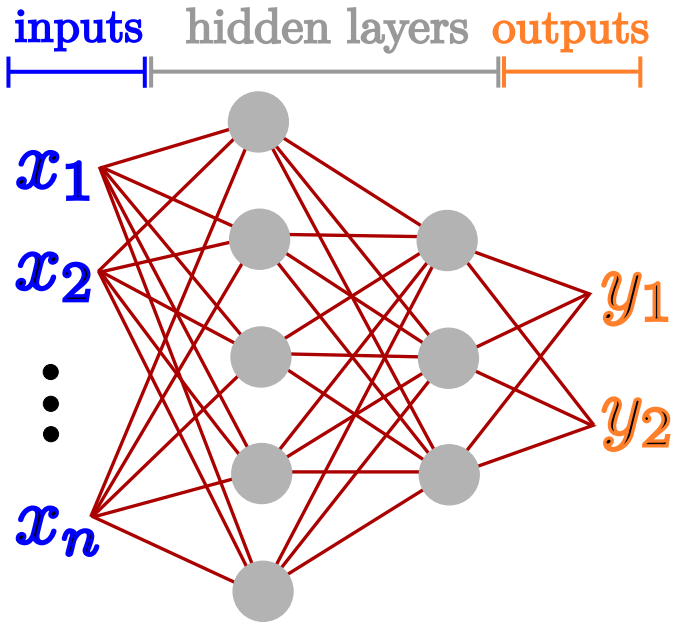
Multi-layer Perceptron (MLP)



- **Universal approximation theorem**[2]: MLP with 1 hidden layer approximates any continuous function

[2] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Math. Control Signal Systems* (1989).

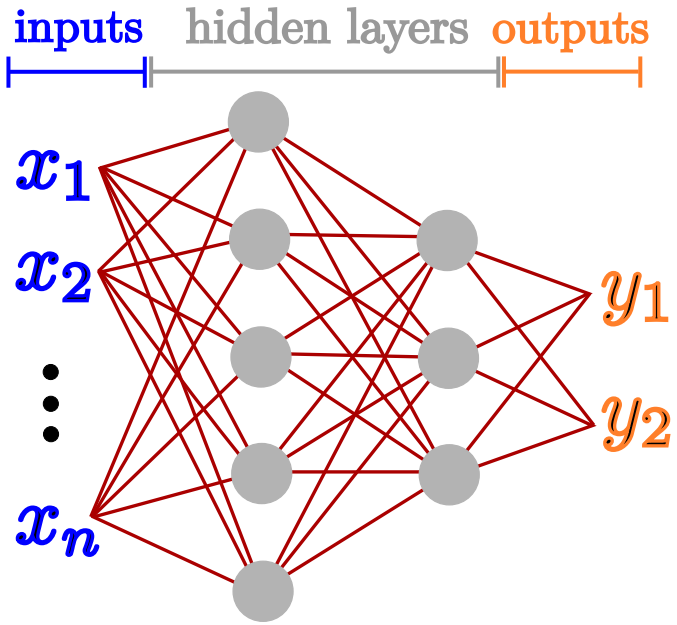
Multi-layer Perceptron (MLP)



- **Universal approximation theorem**[2]: MLP with 1 hidden layer approximates any continuous function ⟶ in theory

[2] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Math. Control Signal Systems* (1989).

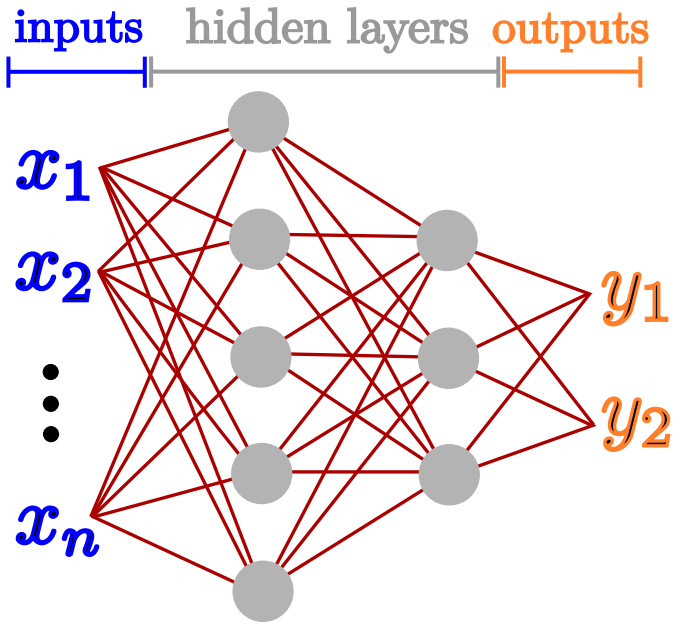
Multi-layer Perceptron (MLP)



- **Universal approximation theorem**[2]: MLP with 1 hidden layer approximates any continuous function ⟶ in theory
- … But is an MLP really the best choice for **every task**?

[2] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Math. Control Signal Systems* (1989).

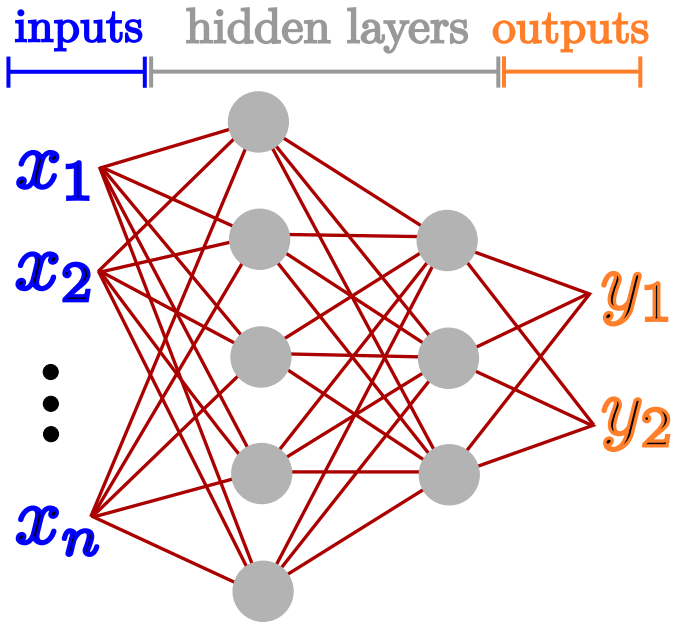
Multi-layer Perceptron (MLP)



- **Universal approximation theorem**[2]: MLP with 1 hidden layer approximates any continuous function ⟶ in theory
- … But is an MLP really the best choice for **every task**?
  - Common issues: data-hungry, poor generalization, black-box

[2] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Math. Control Signal Systems* (1989).

Multi-layer Perceptron (MLP)



- **Universal approximation theorem**[2]: MLP with 1 hidden layer approximates any continuous function ⟶ in theory
- … But is an MLP really the best choice for **every task**?
  - Common issues: data-hungry, poor generalization, black-box
  - The **internal architecture** of a NN matters
    ⟶ Try this: Tensorflow Playground

[2] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Math. Control Signal Systems* (1989).

Literature Examples

- *Computer vision*: **convolutional neural networks**, embedding the concept of spatial invariance[3]

[3] Yann LeCun, Koray Kavukcuoglu, and Clement Farabet. "Convolutional networks and applications in vision". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 2010.

[4] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* (1997).

[5] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

The Internal Architecture Matters

Literature Examples

- *Computer vision*: **convolutional neural networks**, embedding the concept of spatial invariance[3]
- *Speech recognition*: **long short-term memory networks**, embedding the concept of short- and long-term memory[4]

[3] Yann LeCun, Koray Kavukcuoglu, and Clement Farabet. "Convolutional networks and applications in vision". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 2010.

[4] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* (1997).

[5] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

Literature Examples

- *Computer vision*: **convolutional neural networks**, embedding the concept of spatial invariance[3]
- *Speech recognition*: **long short-term memory networks**, embedding the concept of short- and long-term memory[4]
- *Natural language processing*: **transformers** embed the concept of self-attention[5] (example: chatGPT)

[3] Yann LeCun, Koray Kavukcuoglu, and Clement Farabet. "Convolutional networks and applications in vision". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 2010.

[4] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* (1997).

[5] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

The Internal Architecture Matters

Literature Examples

- *Computer vision*: **convolutional neural networks**, embedding the concept of spatial invariance[3]
- *Speech recognition*: **long short-term memory networks**, embedding the concept of short- and long-term memory[4]
- *Natural language processing*: **transformers** embed the concept of self-attention[5] (example: chatGPT)

Can we devise **new NN architectures**
to model/control **physical systems**? (e.g. robots)

[3] Yann LeCun, Koray Kavukcuoglu, and Clement Farabet. "Convolutional networks and applications in vision". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 2010.

[4] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* (1997).

[5] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

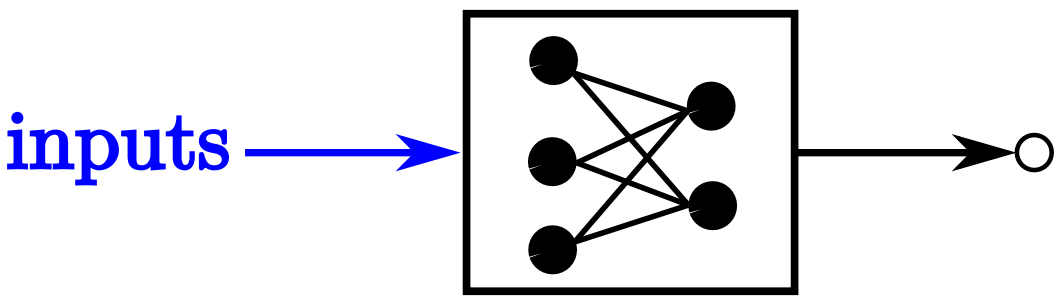
Motivation

Physics-Guided Learning

Physics-Guided NN Approaches

How to **embed physics**- or model-based knowledge in NNs?

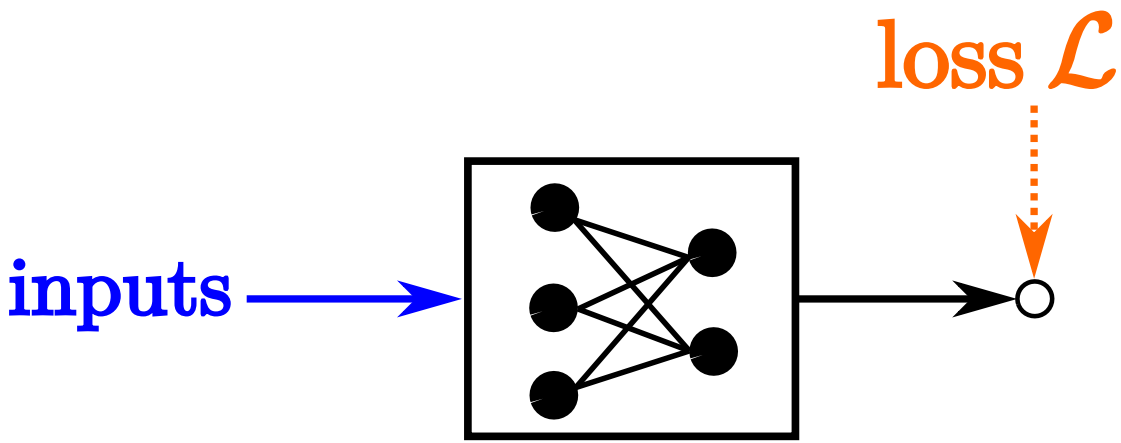
- Curate the training data or **NN inputs**



Physics-Guided NN Approaches

How to **embed physics**- or model-based knowledge in NNs?

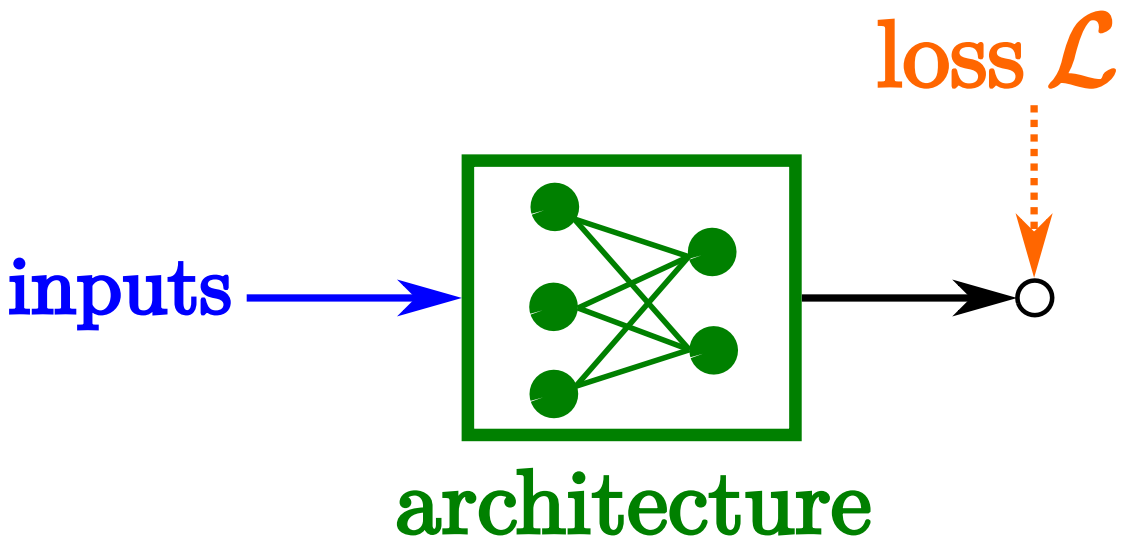
- Curate the training data or **NN inputs**
- Design a **loss function** $\mathcal{L}$



Physics-Guided NN Approaches

How to **embed physics**- or model-based knowledge in NNs?

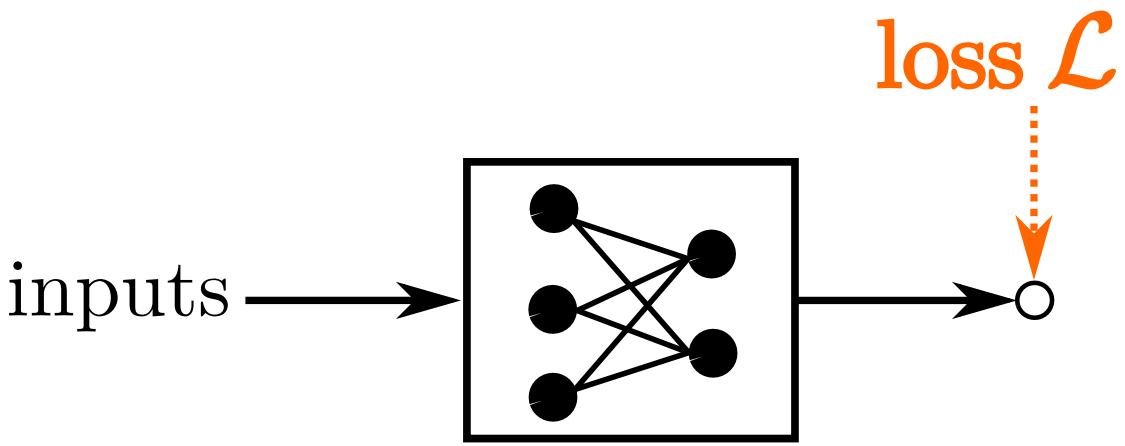
- Curate the training data or **NN inputs**
- Design a **loss function** $\mathcal{L}$
- Design the **internal architecture**



Physics-Guided NN Approaches

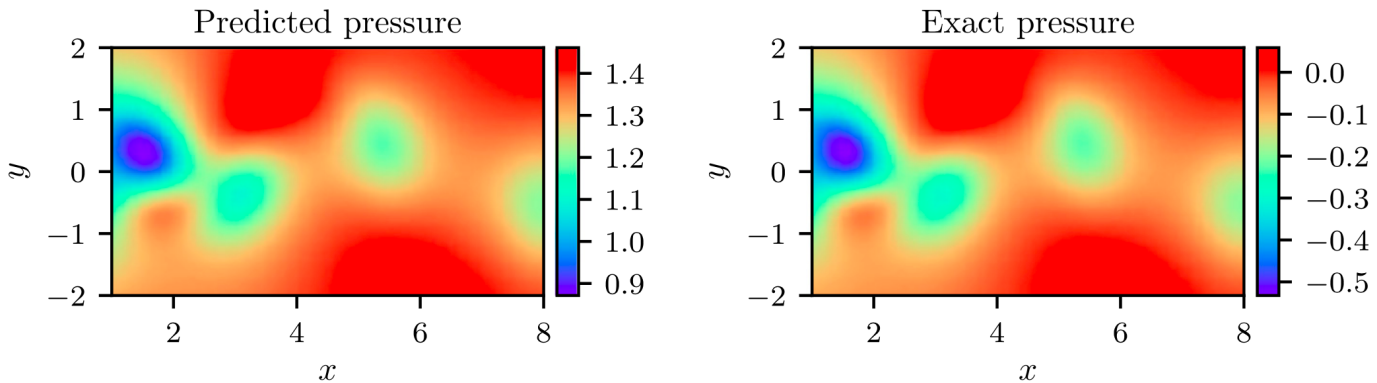
How to **embed physics**- or model-based knowledge in NNs?

- Curate the training data or NN inputs
- Design a **loss function** $\mathcal{L}$ $\rightarrow$ **Physics-informed NNs** (PINNs)
- Design the internal architecture



Physics-informed NNs (PINNs), Raissi et at. (2019)

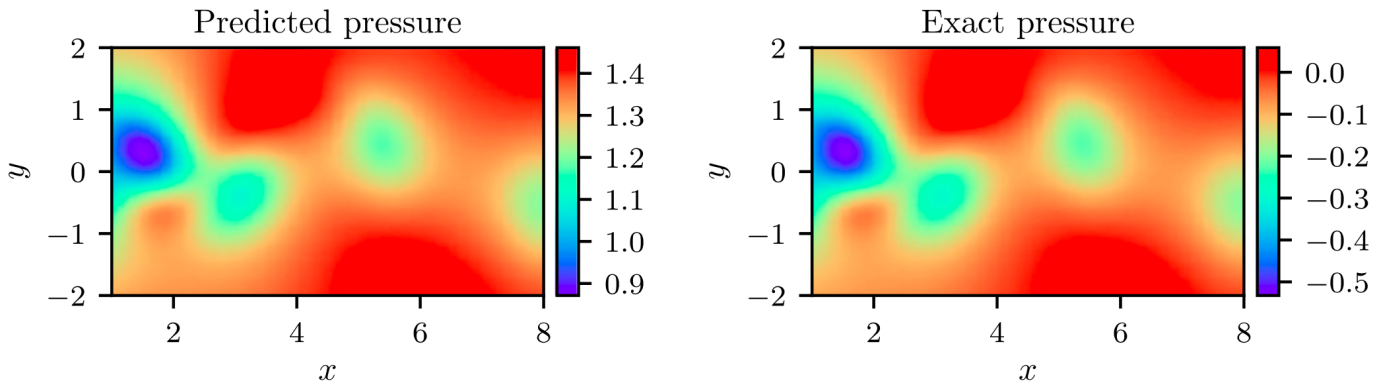
- **Idea**: Data-driven solution or discovery of complex PDEs



[6] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* (2019).

Physics-informed NNs (PINNs), Raissi et at. (2019)

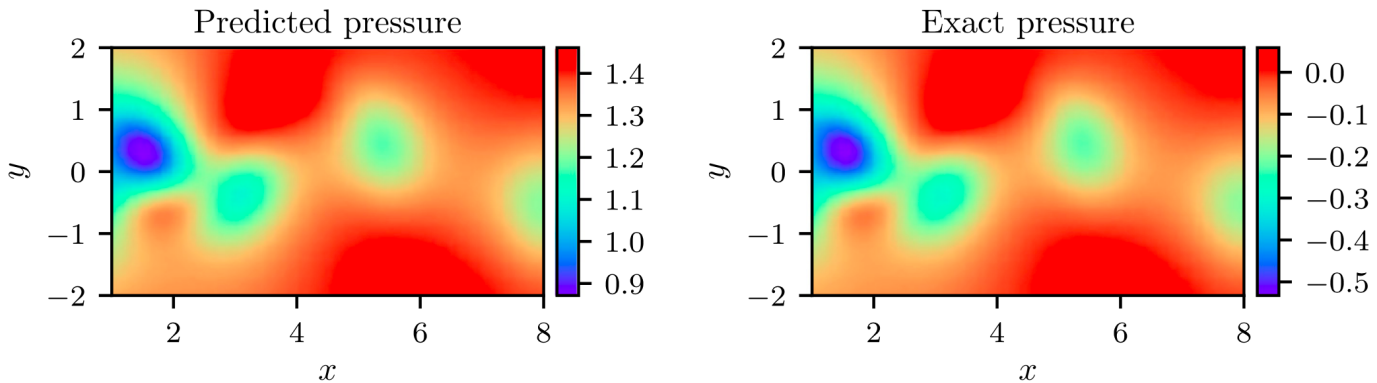
- **Idea**: Data-driven solution or discovery of complex PDEs
- **How**: ***Structured loss function*** based on the PDE residuals, but using deep *unstructured* NNs



[6]M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* (2019).

Physics-informed NNs (PINNs), Raissi et at. (2019)

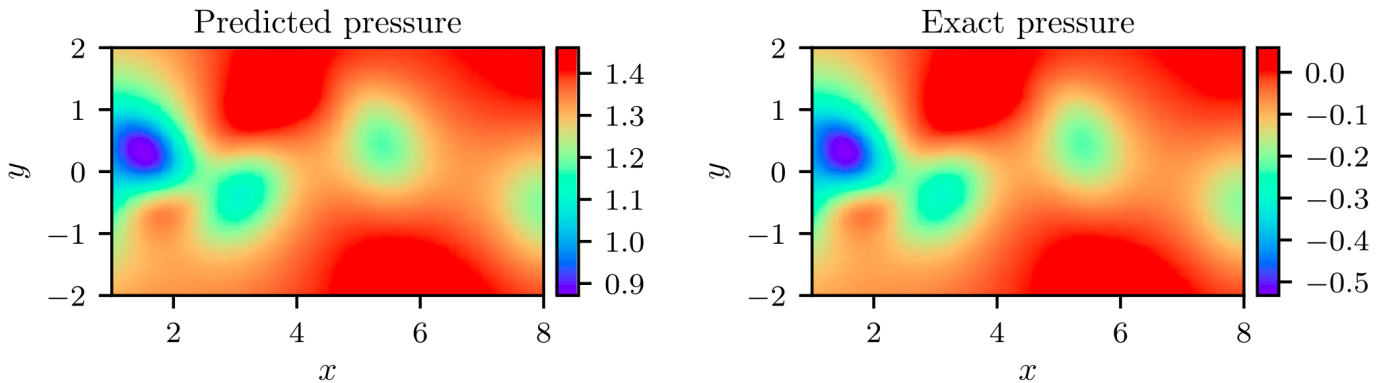
- **Idea**: Data-driven solution or discovery of complex PDEs
- **How**: ***Structured loss function*** based on the PDE residuals, but using deep *unstructured* NNs
- **Pros**: Learn the solution of PDEs using small training datasets; accelerate simulations of complex systems



[6] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* (2019).

Physics-informed NNs (PINNs), Raissi et at. (2019)

- **Idea**: Data-driven solution or discovery of complex PDEs
- **How**: ***Structured loss function*** based on the PDE residuals, but using deep *unstructured* NNs
- **Pros**: Learn the solution of PDEs using small training datasets; accelerate simulations of complex systems
- **Examples**: Fluid dynamics (Navier Stokes), quantum mechanics, solid mechanics, reaction-diffusion[6]. **Note**: no modeling of real physical systems, only PDEs



[6]M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* (2019).

Other approaches related to PINNs

- **Hamiltonian**[7] and **Lagrangian NNs**[8] that learn to conserve energy in mechanical systems and robotic tasks
- **Pontryagin AI**[9] to learn the policies of optimal control tracking problems (OCPs):
  - NN to approximate the controls of an OCP
  - NN trained with a tracking loss function
  - No need to solve the adjoint equations of the OCP
- NNs to approximate the value function of Hamilton-Jacobi-Bellman equations for **dynamic programming**[10]

---

[7] Sam Greydanus, Misko Dzamba, and Jason Yosinski. *Hamiltonian Neural Networks*. 2019.

[8] Michael Lutter, Kim Listmann, and Jan Peters. "Deep Lagrangian Networks for end-to-end learning of energy-based control for under-actuated systems". In: 2019.

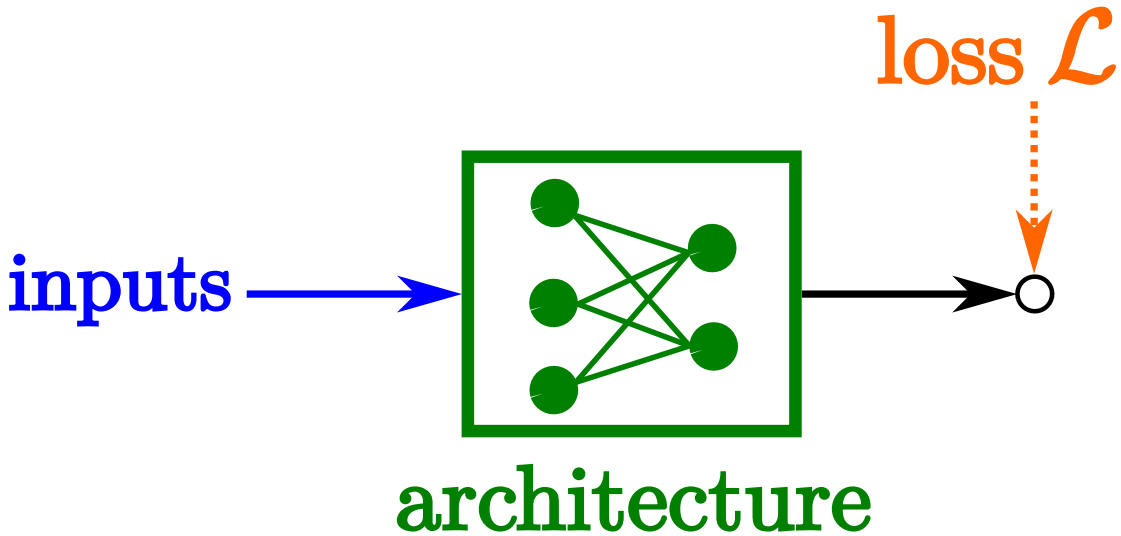
[9] Lucas Böttcher, Nino Antulov-Fantulin, and Thomas Asikis. "AI Pontryagin or how artificial neural networks learn to control dynamical systems". In: *Nature Communications* (2022).

[10] Murad Abu-Khalaf and Frank L. Lewis. "Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach". In: *Automatica* (2005).

Physics-Guided NN Approaches

How to **embed physics**- or model-based knowledge in NNs?

- Curate the training data or **NN inputs**
- Design a **loss function** $\mathcal{L}$
- Design the **internal architecture**

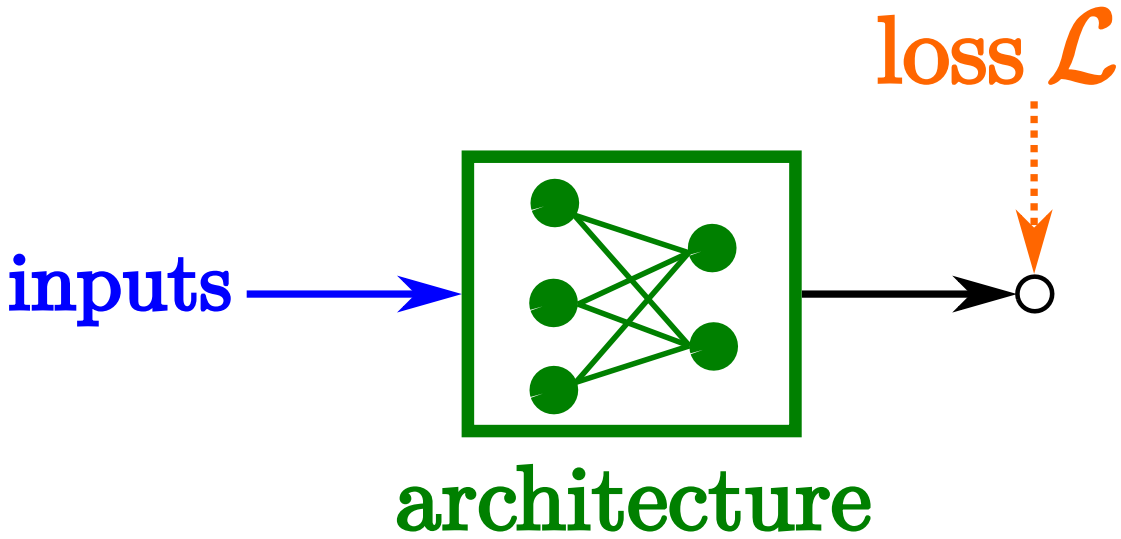


Physics-Guided NN Approaches

How to **embed physics**- or model-based knowledge in NNs?

- Curate the training data or **NN inputs**
- Design a **loss function** $\mathscr{L}$
- Design the **internal architecture**

  → **Model-Structured NNs** (MS-NNs)



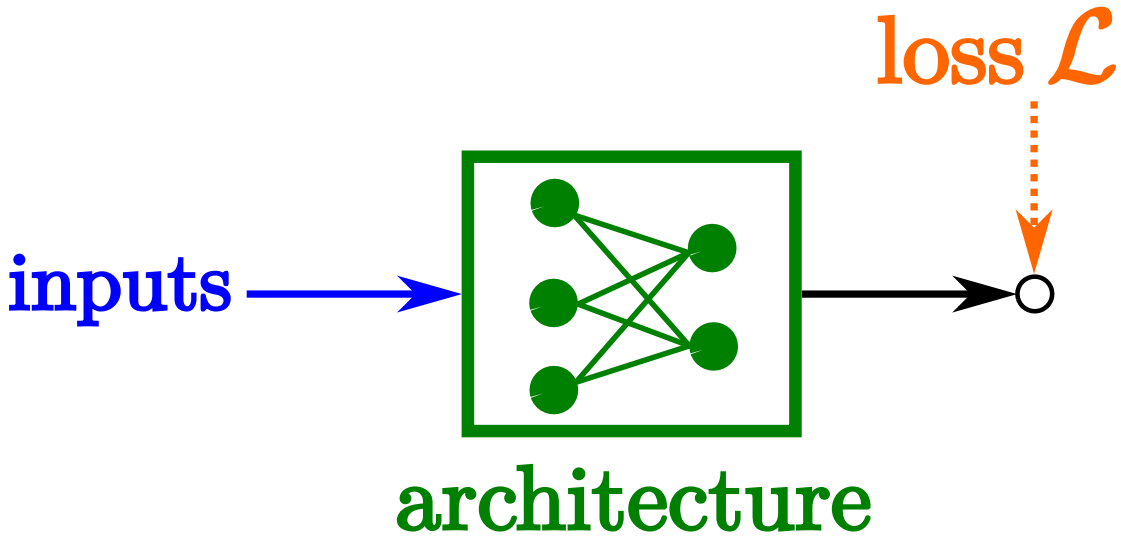
Physics-Guided NN Approaches

How to **embed physics**- or model-based knowledge in NNs?

- Curate the training data or **NN inputs**
- Design a **loss function** $\mathscr{L}$
- Design the **internal architecture**

} **NNodely**

→ **Model-Structured NNs** (MS-NNs)



NNodely

NNodely

Model-Structured Neural Networks (MS-NNs)

Model-Structured Neural Networks (MS-NNs)

Idea

Embed domain-specific knowledge into the **NN architecture**

Model-Structured Neural Networks (MS-NNs)

Idea

Embed domain-specific knowledge into the **NN architecture**

How

Start from a physics-based model and ***neuralize*** its structure

Model-Structured Neural Networks (MS-NNs)

Idea

Embed domain-specific knowledge into the **NN architecture**

How

Start from a physics-based model and ***neuralize*** its structure

Use

Modeling, *control* and *estimation* of physical systems

Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x,
Output: longitudinal accel. a_x

Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x,

Output: longitudinal accel. $a_x \longrightarrow$ naive multi-layer perc. $a_x = \mathrm{NN}(p, v_x)$

Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x,

Output: longitudinal accel. $a_x \longrightarrow$ ~~naive multi-layer perc. $a_x = \mathrm{NN}(p, v_x)$~~

We actually know the vehicle dynamics!

Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x,
Output: longitudinal accel. a_x

$$a_x = \frac{1}{m} \sum_i F_i$$

Example: Longitudinal Vehicle Dynamics

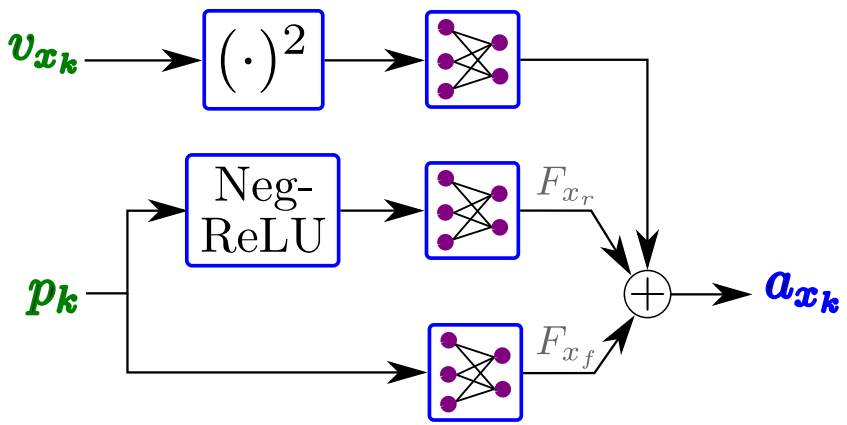
Inputs: pedal $p \in [-1, 1]$, speed v_x,
Output: longitudinal accel. a_x

$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p \quad) + F_{x_r}(p) - k_d \, {v_x}^2 \right]$$

Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x,
Output: longitudinal accel. a_x

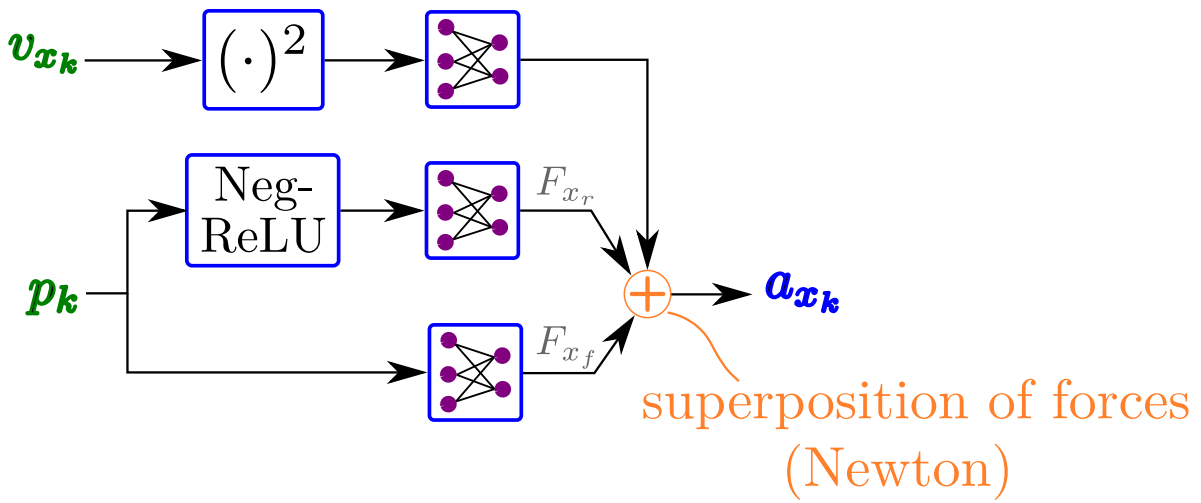
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p \quad) + F_{x_r}(p) - k_d \, {v_x}^2 \right]$$



Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x,
Output: longitudinal accel. a_x

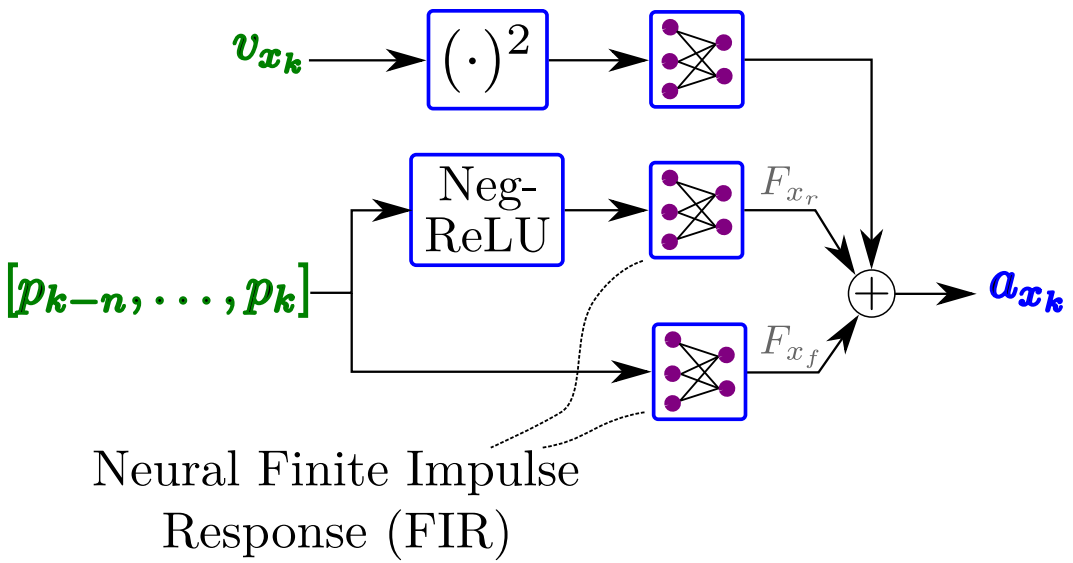
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p \quad) + F_{x_r}(p) - k_d \, {v_x}^2 \right]$$



Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x,
Output: longitudinal accel. a_x

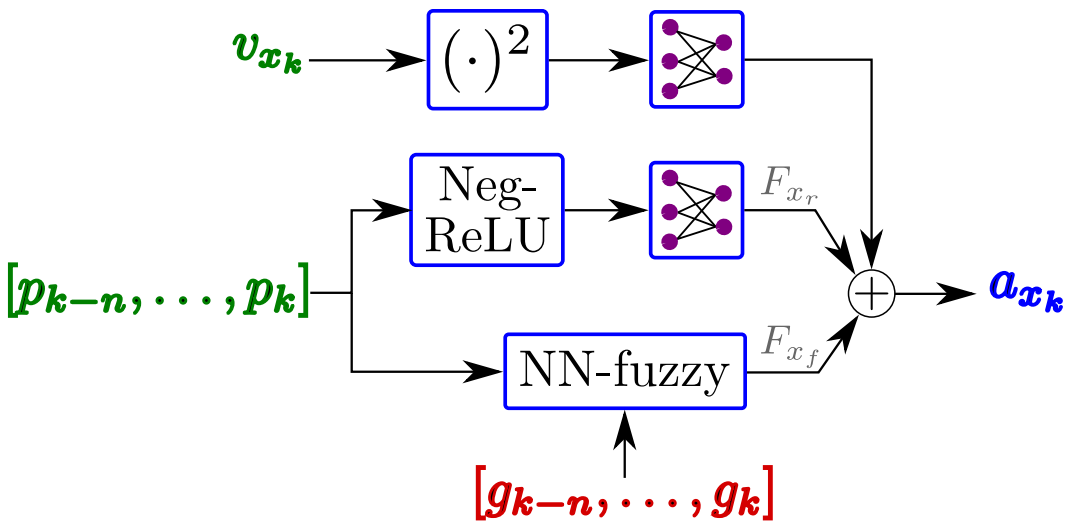
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p \quad) + F_{x_r}(p) - k_d \, {v_x}^2 \right]$$



Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x, gear g,
Output: longitudinal accel. a_x

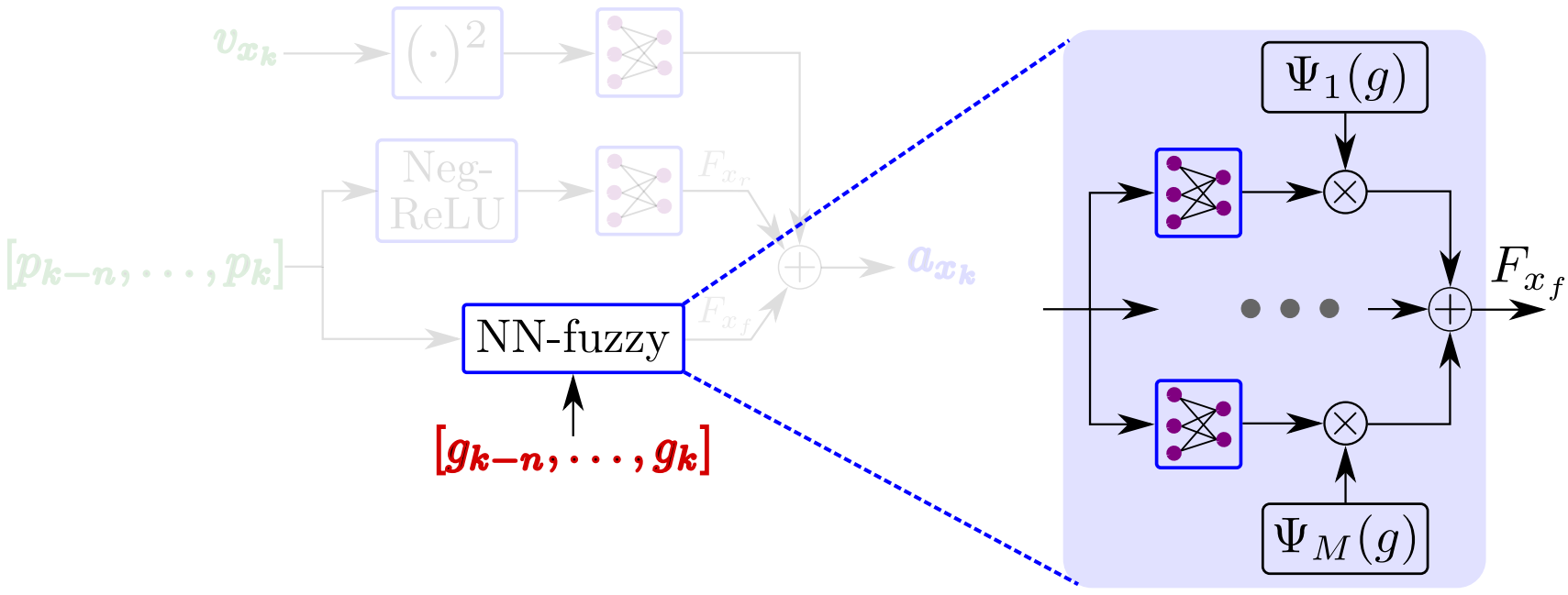
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p, g) + F_{x_r}(p) - k_d \, {v_x}^2 \right]$$



Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x, gear g,
Output: longitudinal accel. a_x

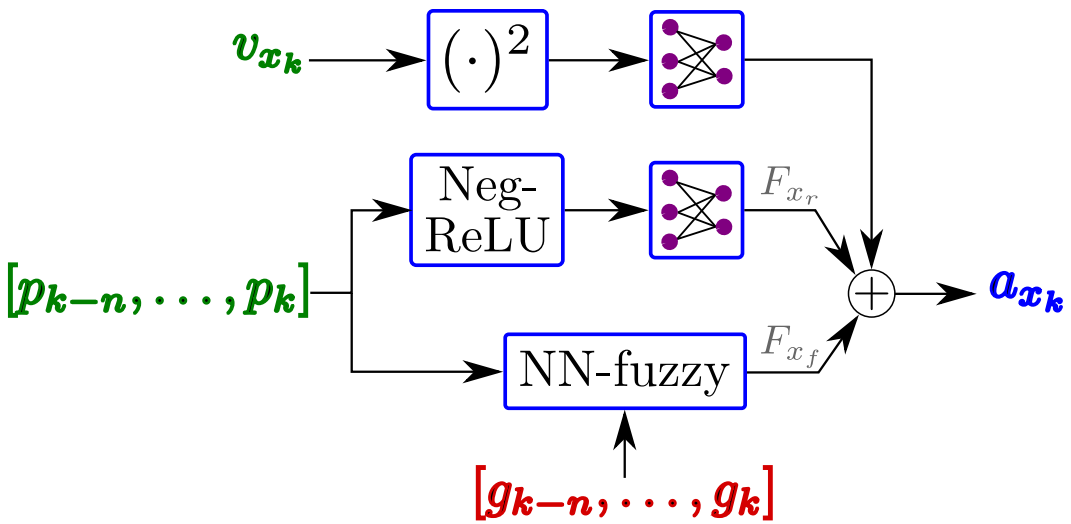
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p, g) + F_{x_r}(p) - k_d \, {v_x}^2 \right]$$



Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x, gear g,
Output: longitudinal accel. a_x

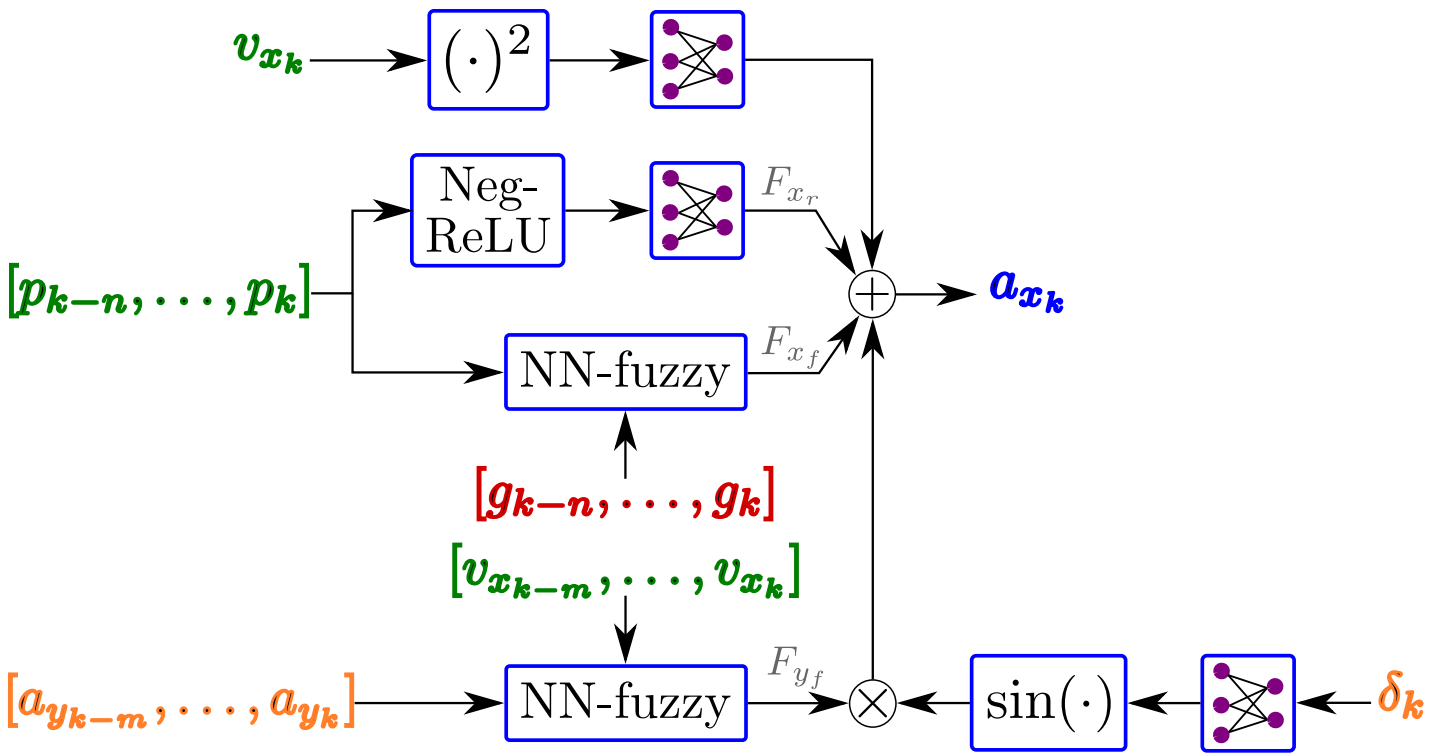
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p, g) + F_{x_r}(p) - k_d \, {v_x}^2 \right]$$



Example: Longitudinal Vehicle Dynamics

Inputs: pedal $p \in [-1, 1]$, speed v_x, gear g, steer δ, lateral accel. a_y
Output: longitudinal accel. a_x

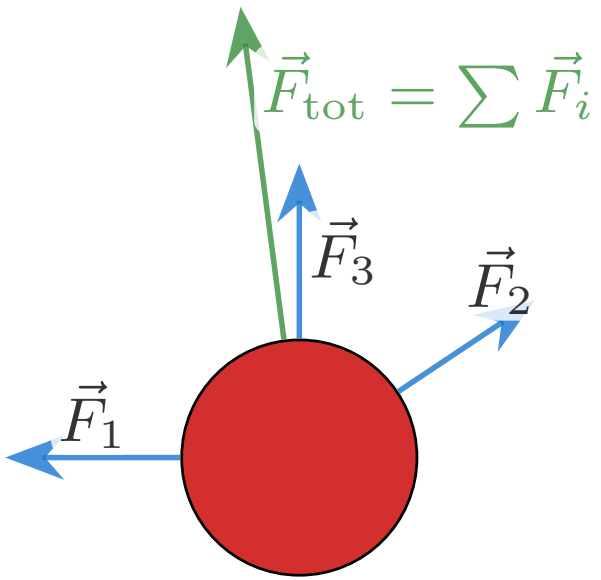
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p, g) + F_{x_r}(p) - k_d \, {v_x}^2 - F_{y_f} \sin(\delta) \right]$$



Embedding Physical Principles in MS-NNs

- Linear **superposition of forces**

$$\dot{x} = f(x,u) = f_x(x) + f_1(x,u_1) + f_2(x,u_2) + \ldots + f_n(x,u_n)$$



Embedding Physical Principles in MS-NNs

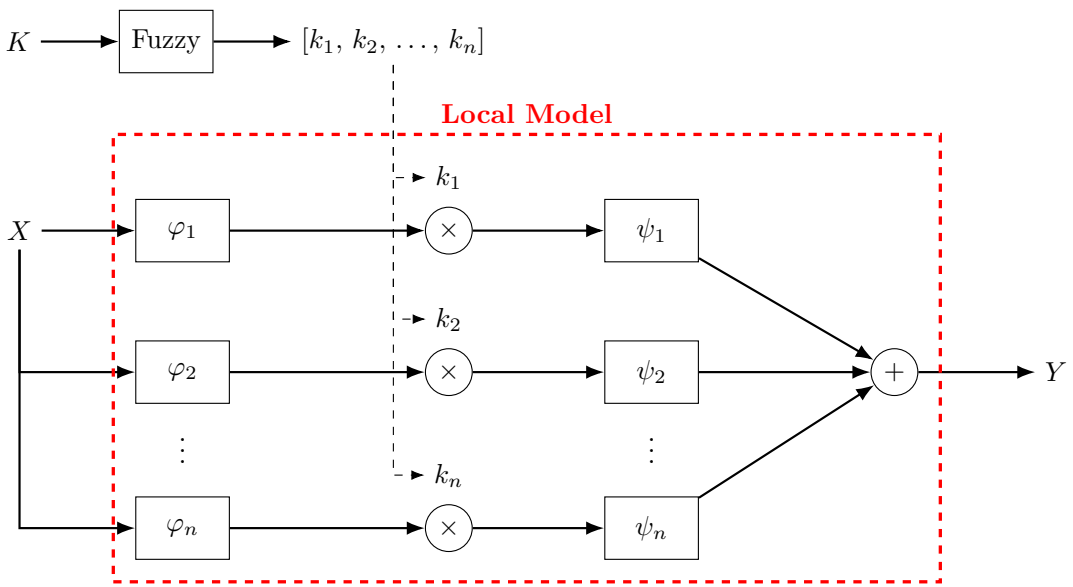
- Linear **superposition of forces**
- Discrete-time system dynamics modeled with finite/infinite impulse response layers (**FIR** or **IIR**)

FIR
$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] = \sum_{i=0}^{N} b_i \cdot x[n-i],$$

IIR
$$y[n] = \sum_{i=0}^{P} b_i x[n-i] + \sum_{i=1}^{Q} a_i y[n-i]$$

Embedding Physical Principles in MS-NNs

- Linear **superposition of forces**
- Discrete-time system dynamics modeled with finite/infinite impulse response layers (**FIR** or **IIR**)
- Unknown nonlinearities addressed with neuro-fuzzy **local models**



Embedding Physical Principles in MS-NNs

- Linear **superposition of forces**
- Discrete-time system dynamics modeled with finite/infinite impulse response layers (**FIR** or **IIR**)
- Unknown nonlinearities addressed with neuro-fuzzy **local models**
- Capturing the residuals of analytical models (such as friction, delays, and backlash) through **ad-hoc neuralization** with **custom funct.**

Embedding Physical Principles in MS-NNs

- Linear **superposition of forces**
- Discrete-time system dynamics modeled with finite/infinite impulse response layers (**FIR** or **IIR**)
- Unknown nonlinearities addressed with neuro-fuzzy **local models**
- Capturing the residuals of analytical models (such as friction, delays, and backlash) through **ad-hoc neuralization** with **custom funct.**
- Embedding the concept of **time** inside the NN

Embedding Physical Principles in MS-NNs

- Linear **superposition of forces**
- Discrete-time system dynamics modeled with finite/infinite impulse response layers (**FIR** or **IIR**)
- Unknown nonlinearities addressed with neuro-fuzzy **local models**
- Capturing the residuals of analytical models (such as friction, delays, and backlash) through **ad-hoc neuralization** with **custom funct.**
- Embedding the concept of **time** inside the NN
- Flexible managing of custom **recursive networks**

NNodely

Framework

Aims

- **Speed-up** the development of MS-NNs w.r.t. standard deep-learning frameworks (e.g., pure PyTorch or TensorFlow)

The code is reduced by **75%**

Aims

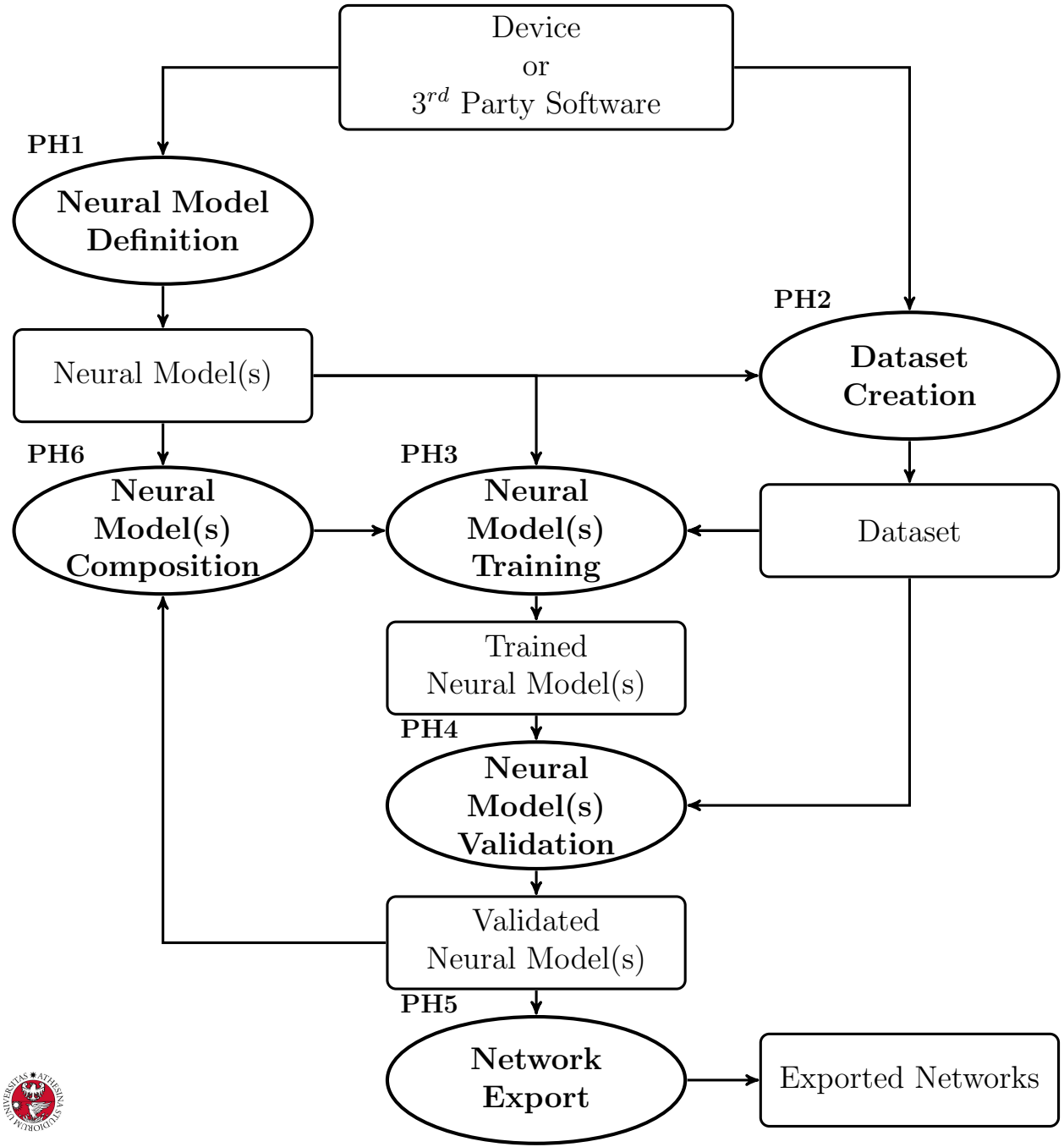
- **Speed-up** the development of MS-NNs w.r.t. standard deep-learning frameworks (e.g., pure PyTorch or TensorFlow)
- **Support professionals** with classical modeling backgrounds, such as physicists and engineers, in developing **physics-guided ML** solutions

Ease of use for **non-experts** in ML

Aims

- **Speed-up** the development of MS-NNs w.r.t. standard deep-learning frameworks (e.g., pure PyTorch or TensorFlow)
- **Support professionals** with classical modeling backgrounds, such as physicists and engineers, in developing **physics-guided ML** solutions
- A **repository** of incremental **know-how** to collect examples of MS-NNs for various applications

Workflow



modely
neuralize your model

Model-Structured Neural Networks with nnodely

MS-NNs Definition (PH1)

- Defining **Inputs**
- Defining **Outputs**
- Defining **Relations**:
  - Aritmetic operations
  - Trigonometric operations
  - Custom functions
  - Local models
  - Fuzzification
  - ...

MS-NN Manipulation

- Data loading and dataset creation (PH2)
- Fitting objectives (PH3)
- Network composition and connection (PH6)
- Training (PH3)
- Validation using model based criteria (PH4)
- MS-NN export (PH5) in **PyTorch** and **ONNX** format

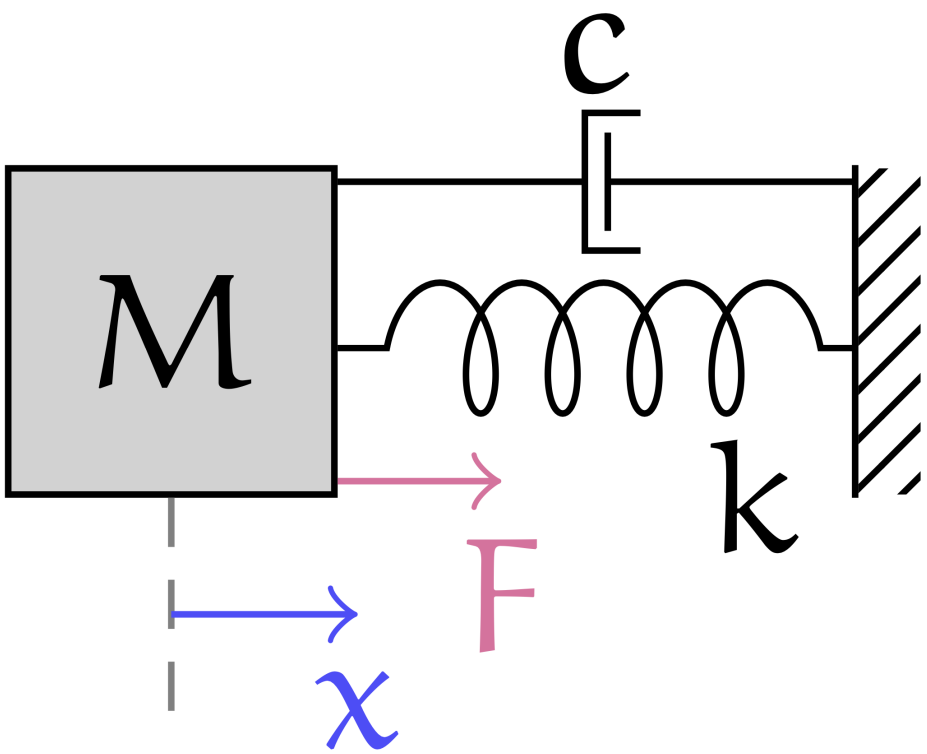
Use Cases

Use Cases

Modeling and Control of a Mass-Spring-Damper System

Mass-Spring-Damper: Modeling

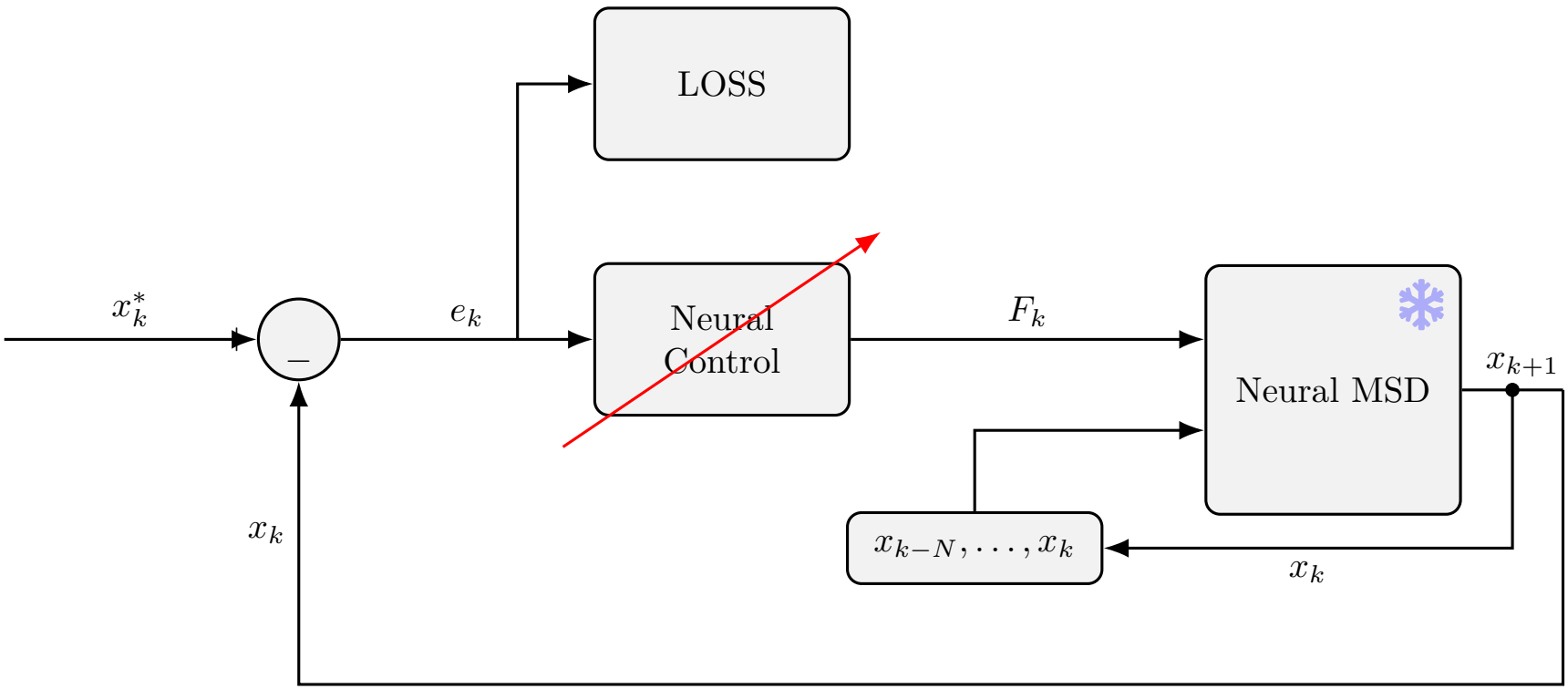
Learn the **dynamics** of a Mass-Spring-Damper system (neural MSD)



$$x_{k+1} = \overbrace{\sum_{k=0}^{N_x-1} x[-k]\cdot h_x[(N_x-1)-k]}^{\mathrm{FIR}(\bar{x})} + \overbrace{F[0]\cdot h_F}^{\mathrm{FIR}(F)}$$

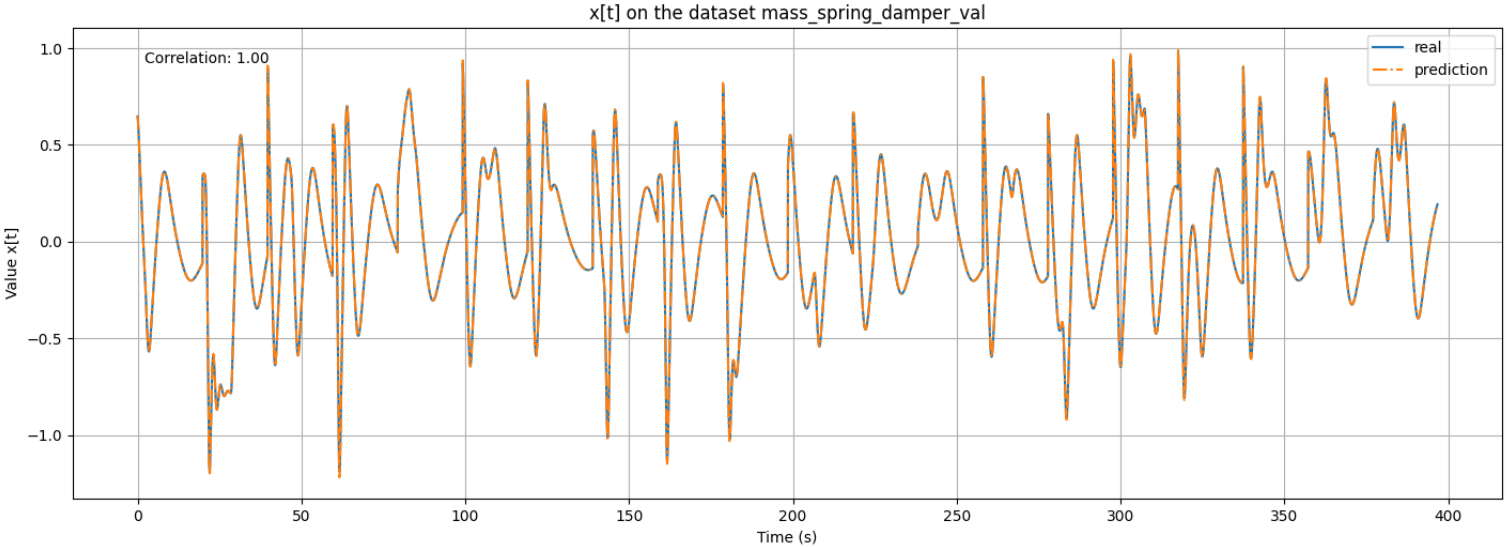
Mass-Spring-Damper: Control

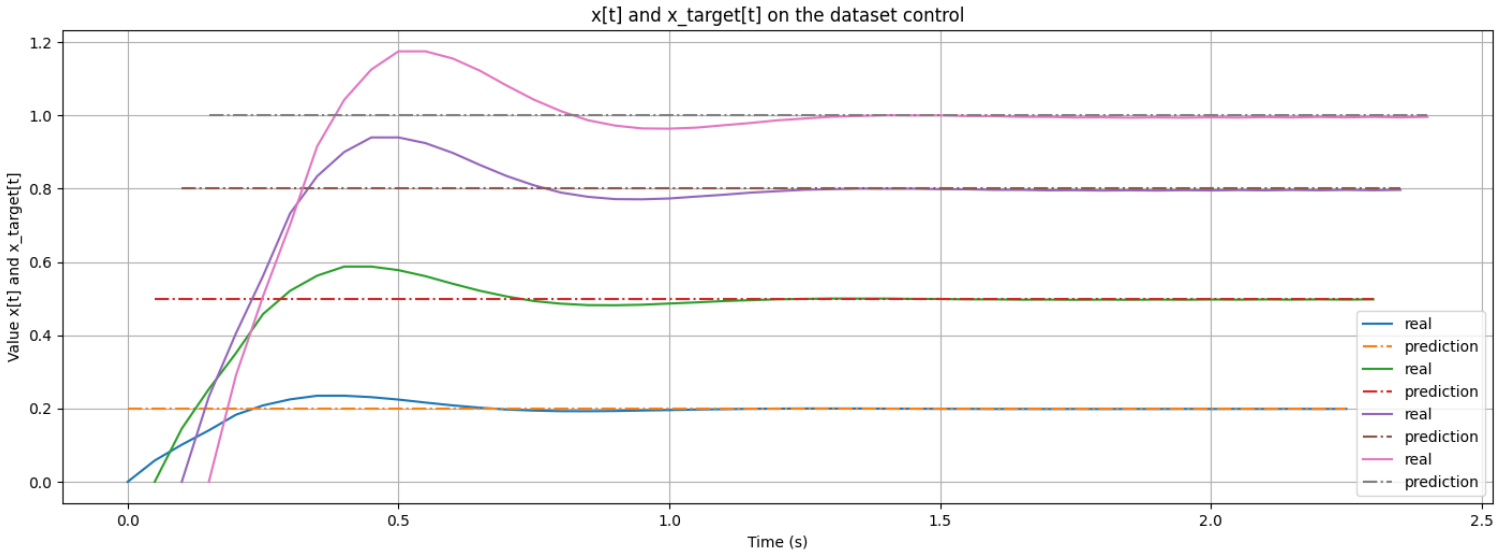
- Learn a **Neural PID** to control a mass-spring-damper system
- Controller gradients are propagated through the learned dynamic model (neural MSD) $\longrightarrow$ **differentiable end-to-end** controller training



$$\text{LOSS} = \sum_{\text{examples}} e_k^2 = \sum (x_k - x_k^*)^2$$

Mass-Spring-Damper: Results for Modelling and Control



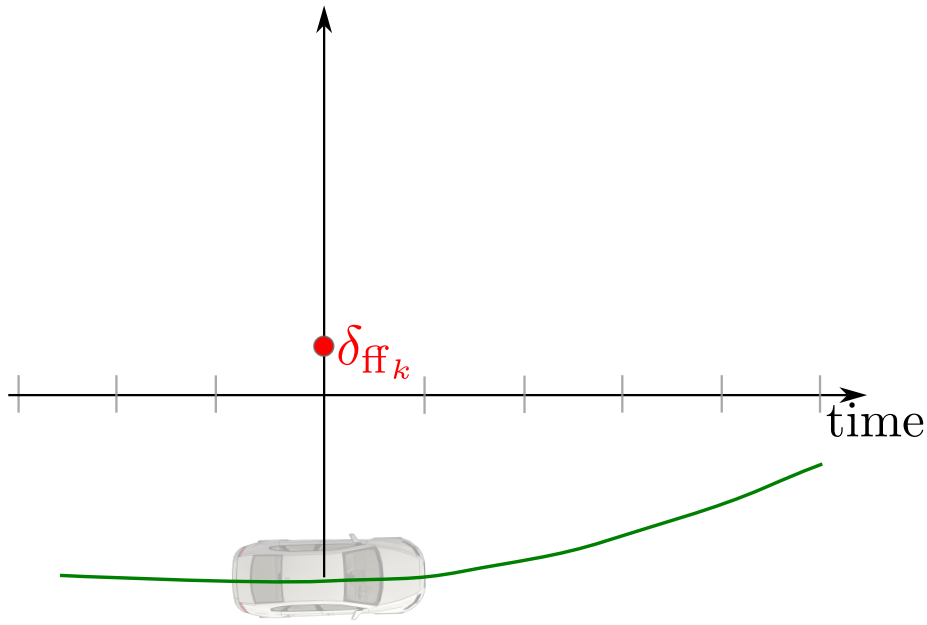


Use Cases

Modeling and Control of the Vehicle Dynamics

Inverse dynamic problem

NN output: steering wheel angle δ_{ff_k}



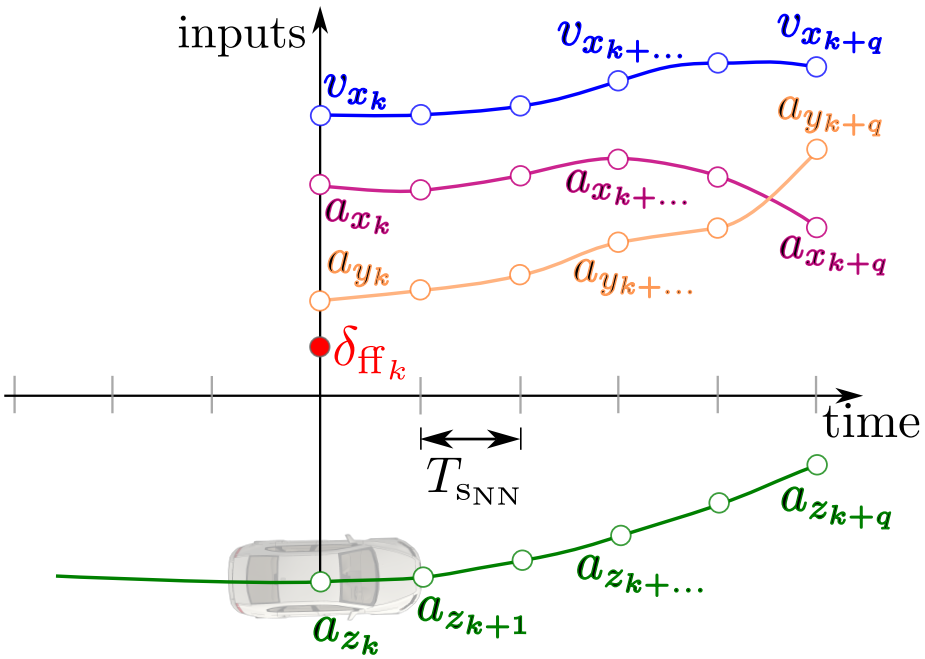
Inverse dynamic problem

NN output: steering wheel angle δ_{ff_k}

NN inputs:

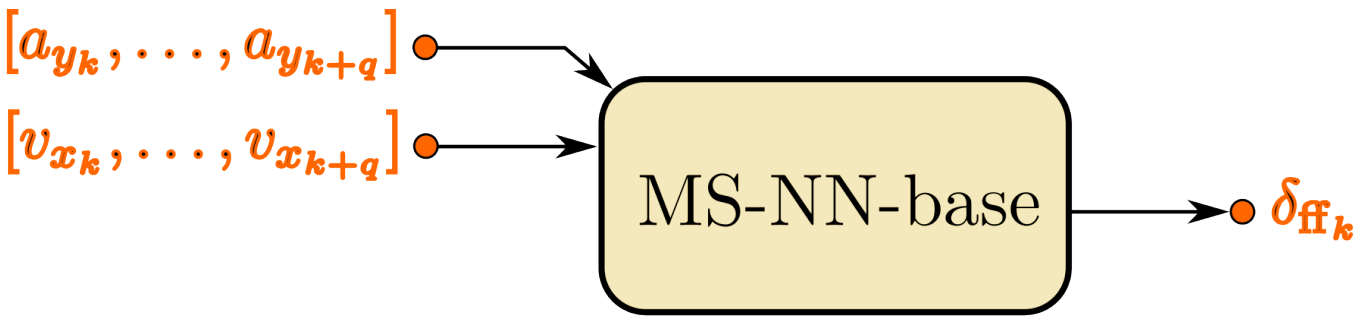
Future reference trajectories: lateral, longit., vertical accelerations and velocity

- $a_{y\,k} = \left[a_{y_k}, \ldots, a_{y_{k+q}}\right]$
- $a_{x\,k} = \left[a_{x_k}, \ldots, a_{x_{k+q}}\right]$
- $a_{z\,k} = \left[a_{z_k}, \ldots, a_{z_{k+q}}\right]$
- $v_{x\,k} = \left[v_{x_k}, \ldots, v_{x_{k+q}}\right]$



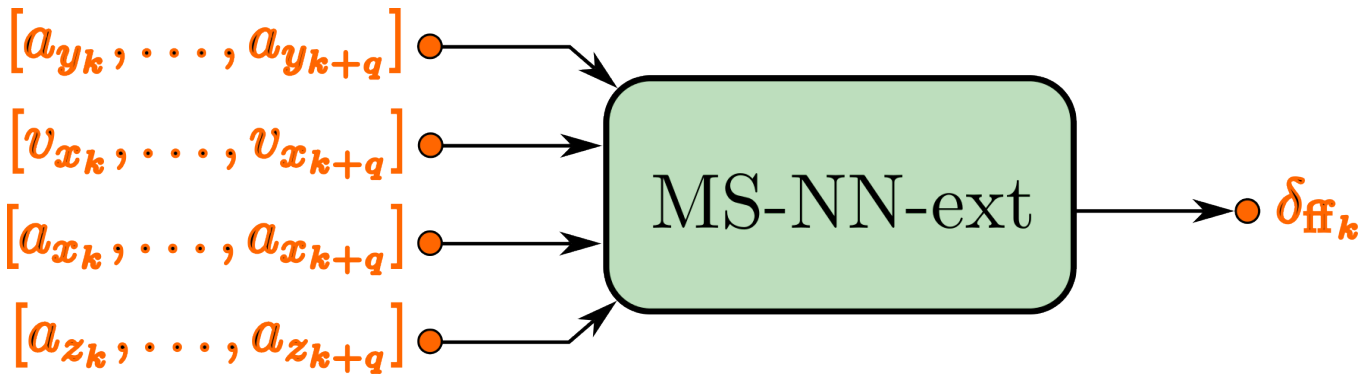
Two incremental variants

- **MS-NN-base** → pure lateral nonlinear dynamics (2D road)



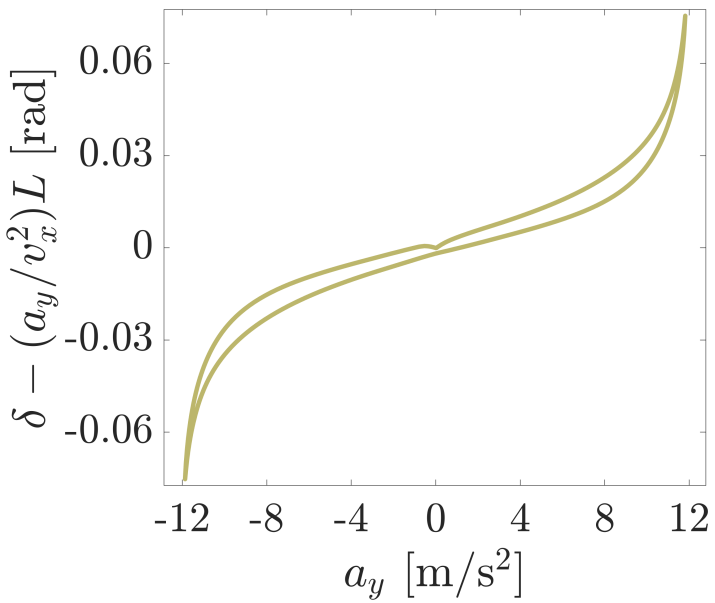
Two incremental variants

- **MS-NN-base** → pure lateral nonlinear dynamics (2D road)
- **MS-NN-ext** → effects of longitudinal and vertical accelerations (3D road)



Steady-state pure lateral dynamics

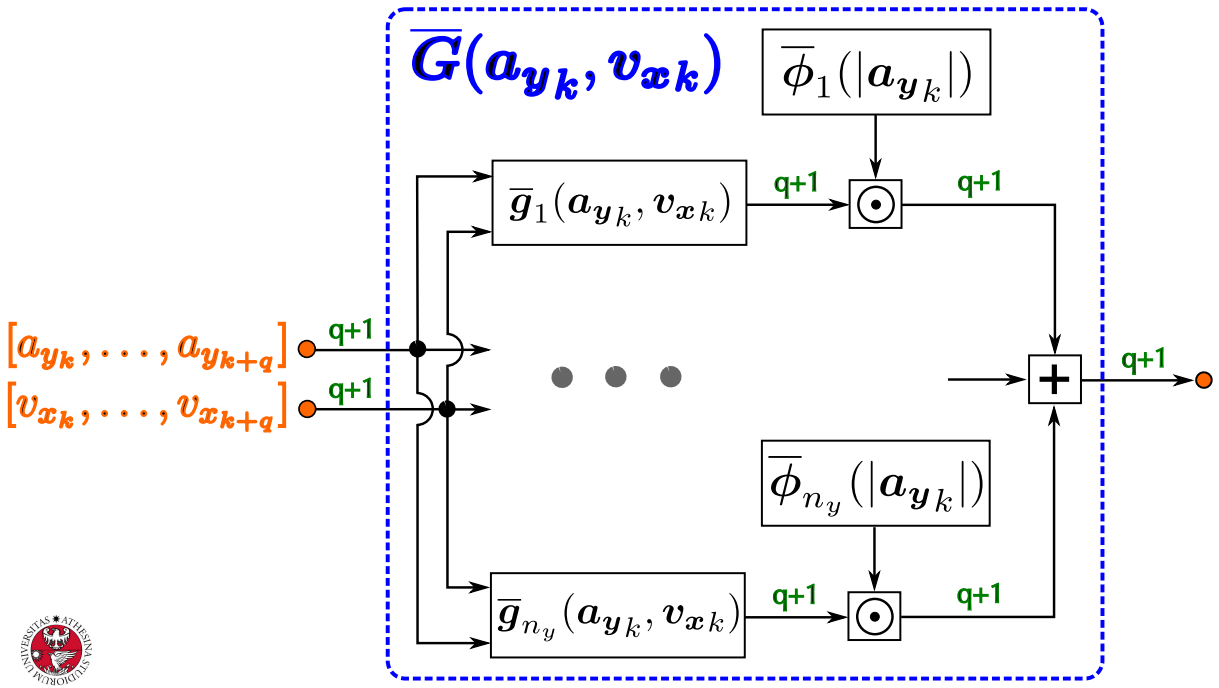
Handling diagram

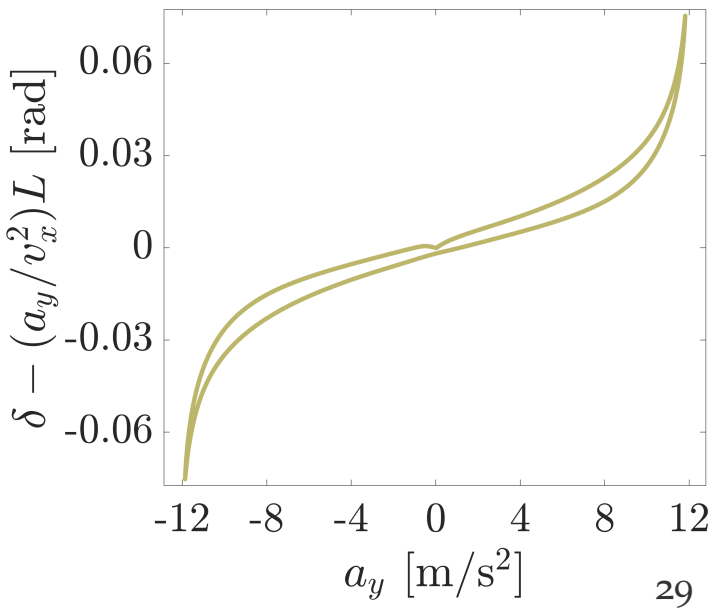


Steady-state pure lateral dynamics

Handling diagram

- **Local neuro-fuzzy** models



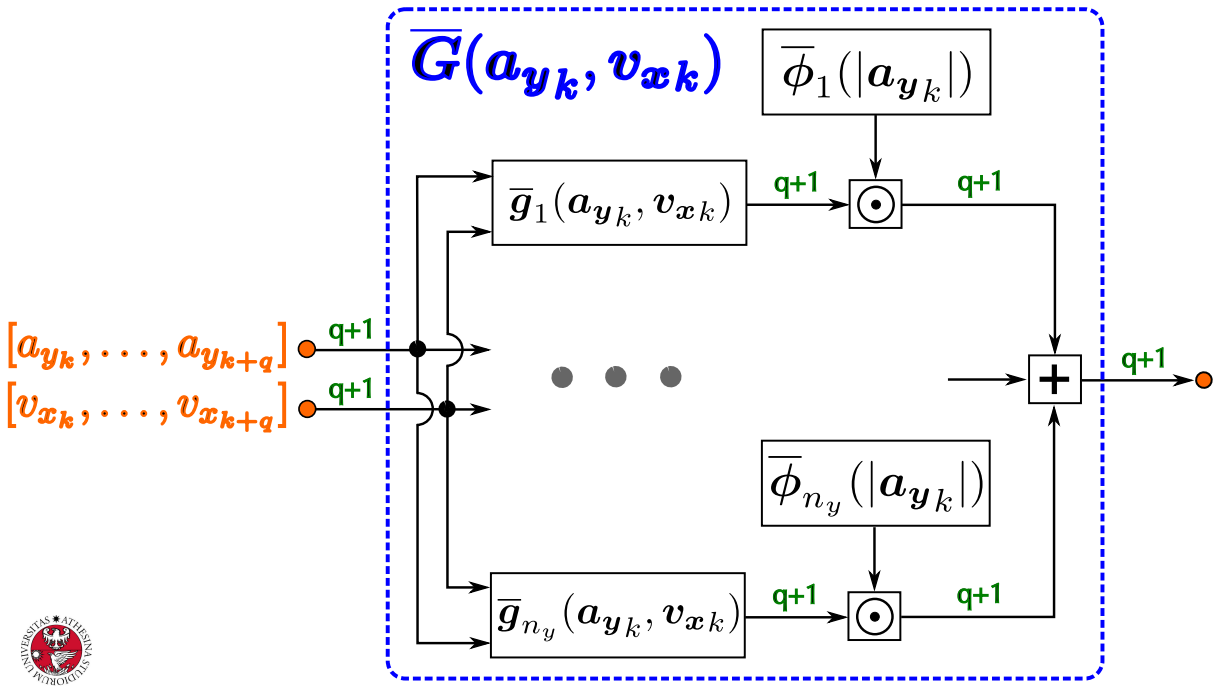


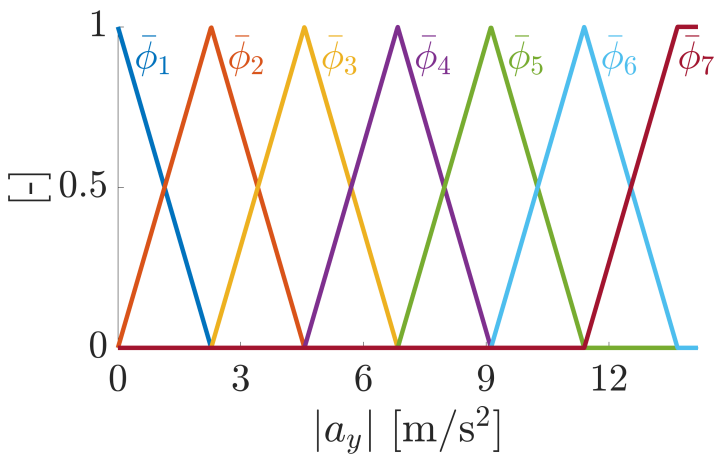
Steady-state pure lateral dynamics

Handling diagram

- **Local neuro-fuzzy** models

$$\bar{g}_i(a_y, v_x) = \frac{a_y}{v_x^2} L + \operatorname{sign}(a_y)\, \mathrm{k}1_i + \left(a_y - a_{y0_i} \operatorname{sign}(a_y)\right) \mathrm{k}2_i$$



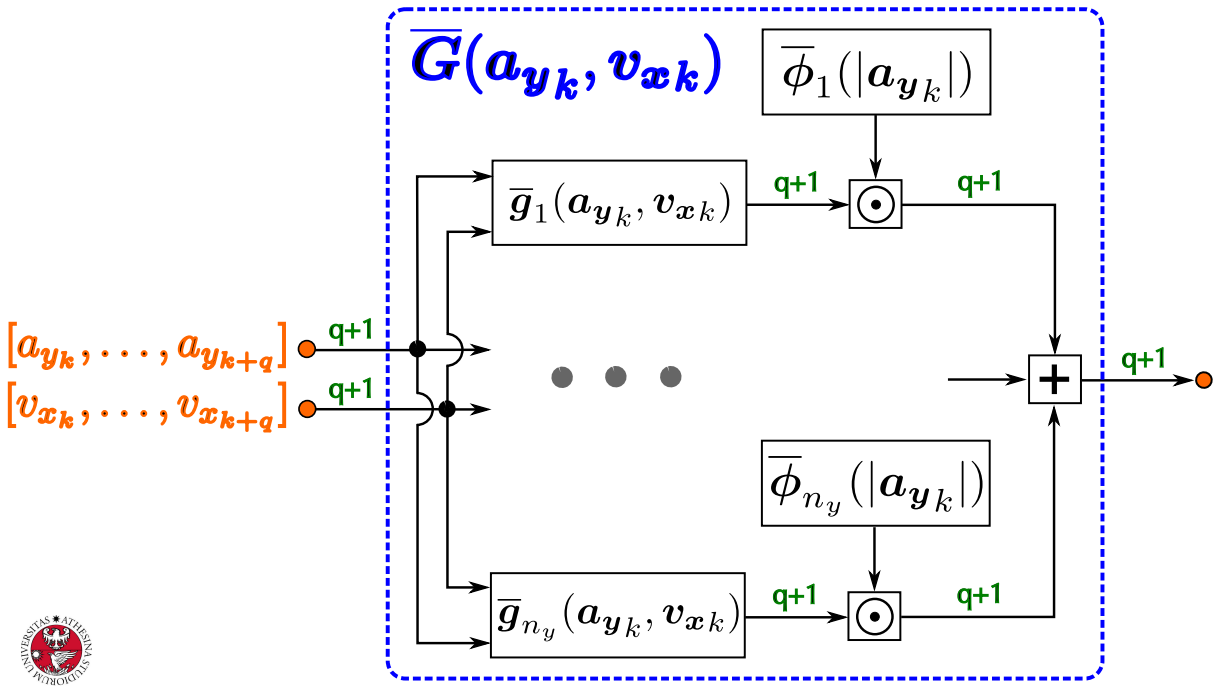


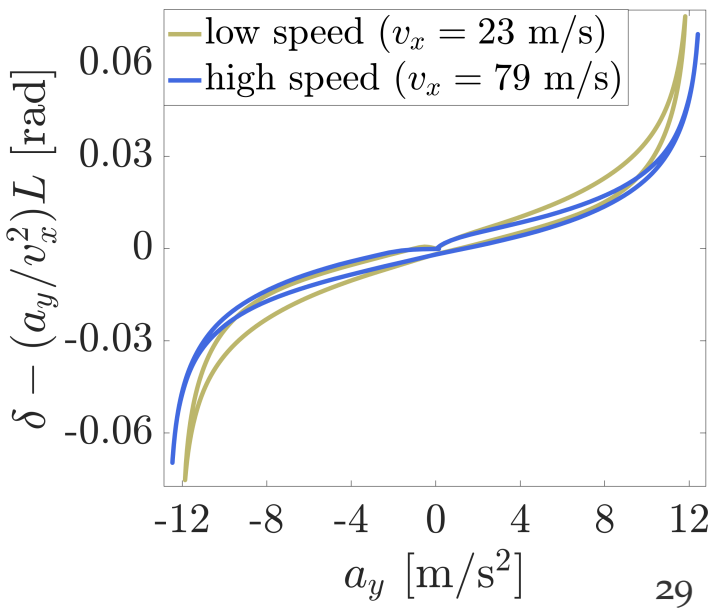
Steady-state pure lateral dynamics

Handling diagram

- **Local neuro-fuzzy** models

$$\bar{g}_i(a_y, v_x) = \frac{a_y}{v_x^2} L + \text{sign}(a_y)\, \text{k}1_i \qquad + \left(a_y - a_{y0_i}\, \text{sign}(a_y)\right) \text{k}2_i$$



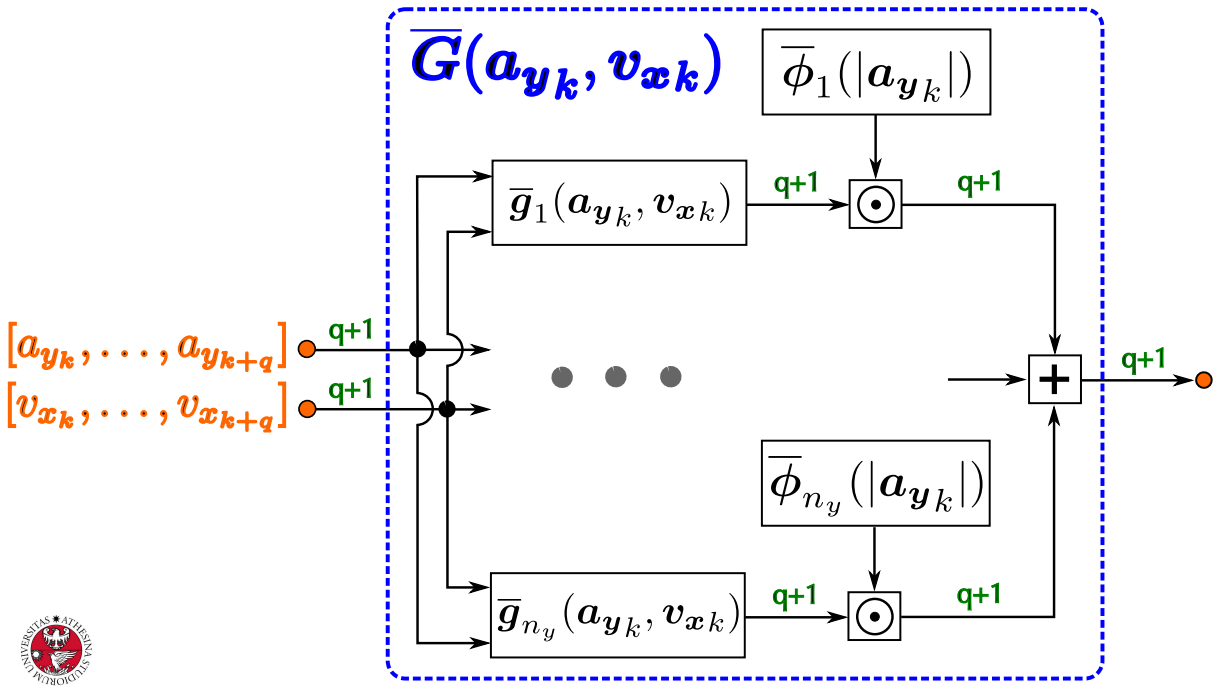


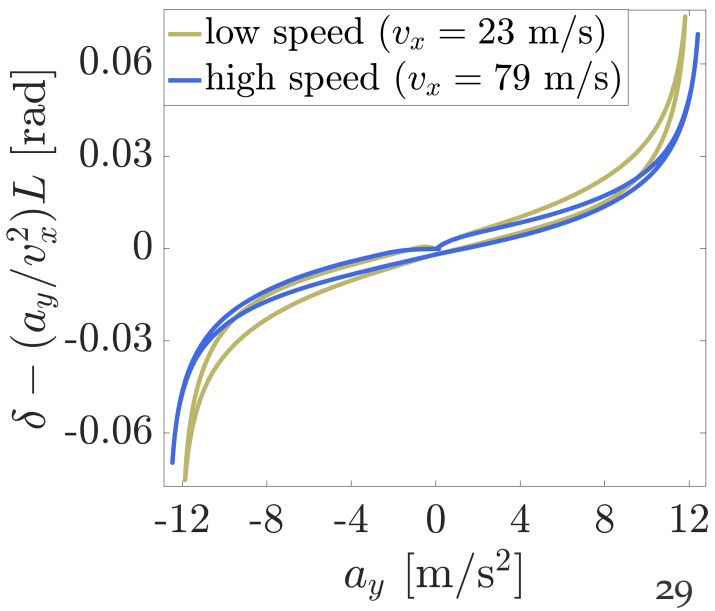
Steady-state pure lateral dynamics

Handling diagram

- **Local neuro-fuzzy** models with v_x-dependency

$$\bar{g}_i(a_y, v_x) = \frac{a_y}{v_x^2} L + \text{sign}(a_y)\, \text{k}1_i(v_x) + \left(a_y - a_{y0_i}\text{sign}(a_y)\right) \text{k}2_i(v_x)$$





Steady-state pure lateral dynamics

Handling diagram

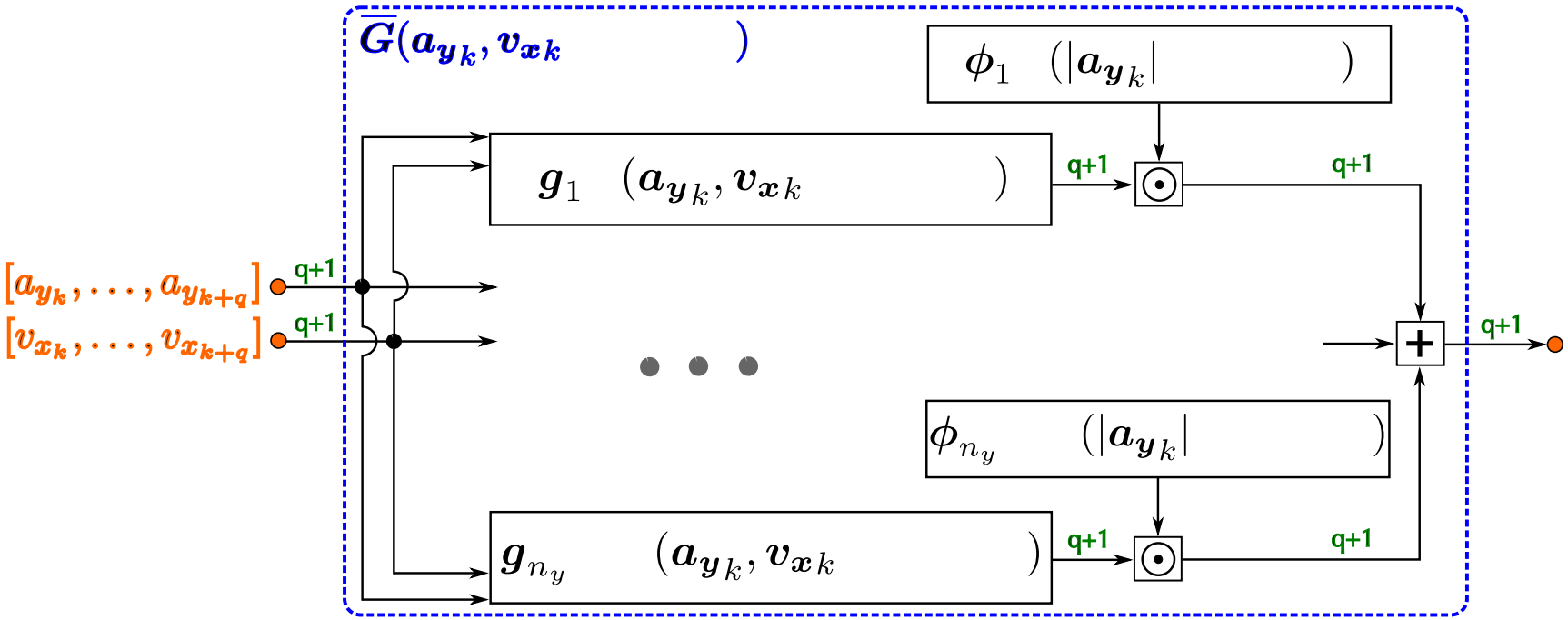
- **Local neuro-fuzzy** models with v_x-dependency

$$\bar{g}_i(a_y, v_x) = \frac{a_y}{v_x^2} L + \text{sign}(a_y)\, \text{k1}_i(v_x) + \left(a_y - a_{y0_i}\, \text{sign}(a_y)\right) \text{k2}_i(v_x)$$

```
def handling_diagr_local(ay,vx,  # inputs
                         ay_0,L,          # constants
                         k1_vx,k2_vx,     # learnable arrays of parameters
                         ):
  sign_ay = torch.sign(ay)  # sign of the lateral acceleration

  # compute k1, as a function of vx
  k1_vx_fun = k1_vx[0,0,0]
  for ii in range(1,k1_vx.size(2)):
    k1_vx_fun = k1_vx_fun + k1_vx[0,0,ii]*torch.pow(vx,ii)

  # compute k2, as a function of vx
  k2_vx_fun = k2_vx[0,0,0]
  for ii in range(1,k2_vx.size(2)):
    k2_vx_fun = k2_vx_fun + k2_vx[0,0,ii]*torch.pow(vx,ii)

  # output of the local model of the handling diagram
  output = (ay/vx**2)*L + k1_vx_fun*sign_ay + k2_vx_fun*(ay - ay_0*sign_ay)
  return output

handling_diagr = ParamFun(handling_diagr_local)

activ_fcns_ay = Fuzzify(centers=chan_centers_ay, functions='Triangular')(ay_abs)

out_local_models_handling_diagr = LocalModel(input_function=handling_diagr_gen, pass_indexes=True)((ay,vx),
                                                                                                     (activ_fcns_ay))
```

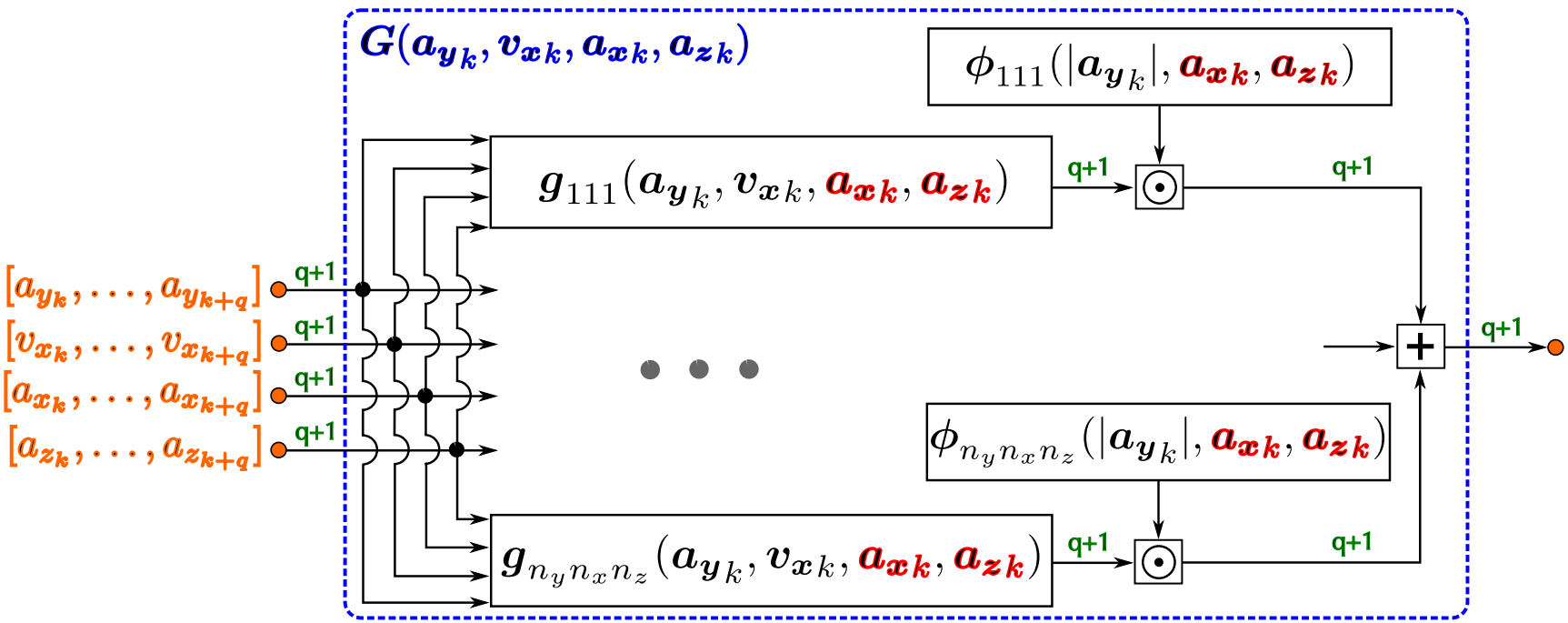
Quasi steady-state combined lateral dynamics

- **Local neuro-fuzzy** models in overlapping regions of a_y



Quasi steady-state combined lateral dynamics

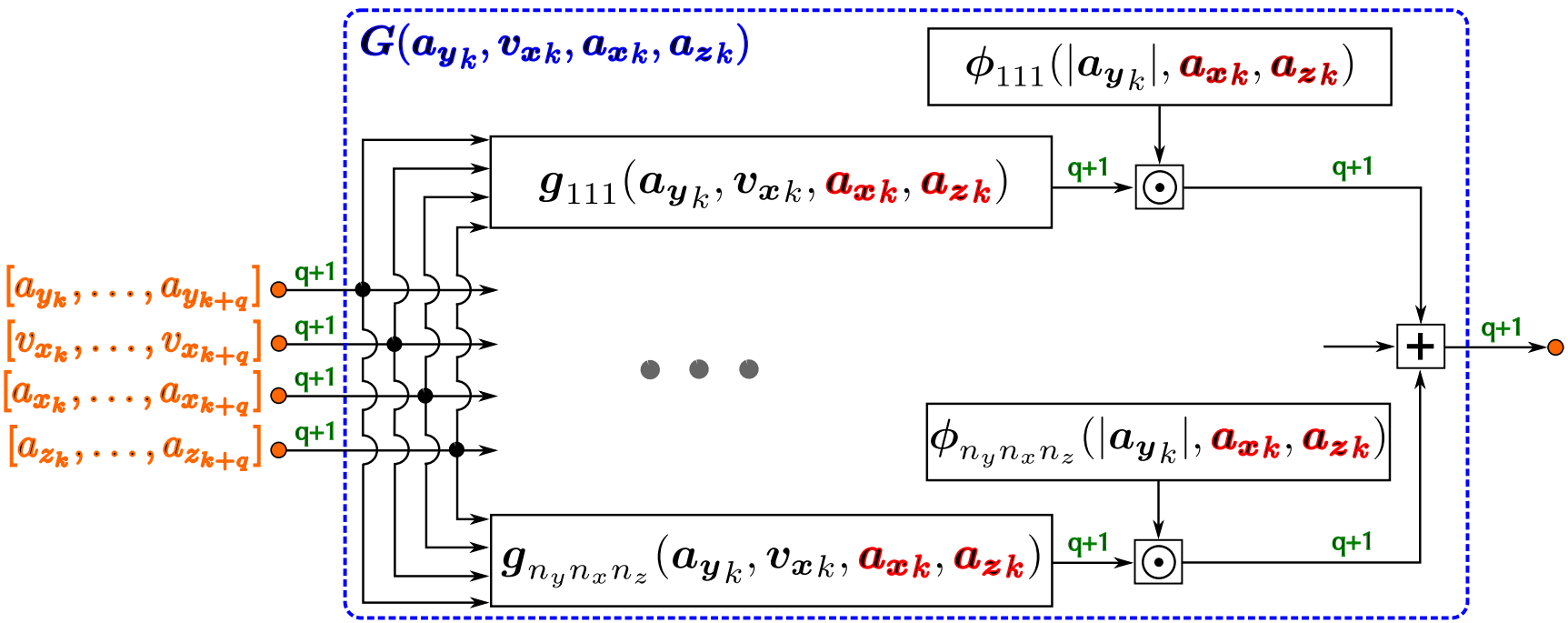
- **Local neuro-fuzzy** models in overlapping regions of a_y, a_x, a_z



Quasi steady-state combined lateral dynamics

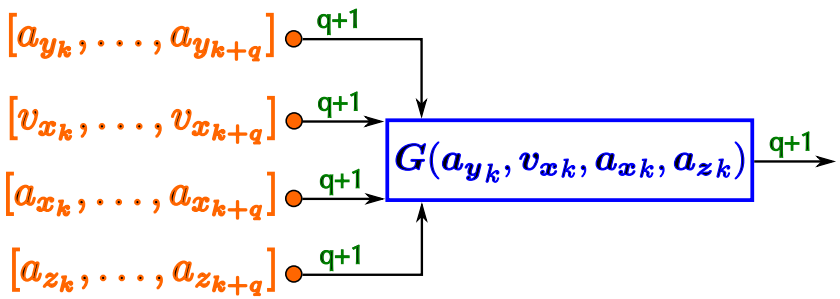
- **Local neuro-fuzzy** models in overlapping regions of a_y, a_x, a_z

  $g_{ijk}(\cdot) \rightarrow$ local approximation of a nonlinear double-track model around $\{a_{y0_i}, a_{x0_j}, a_{z0_k}\}$



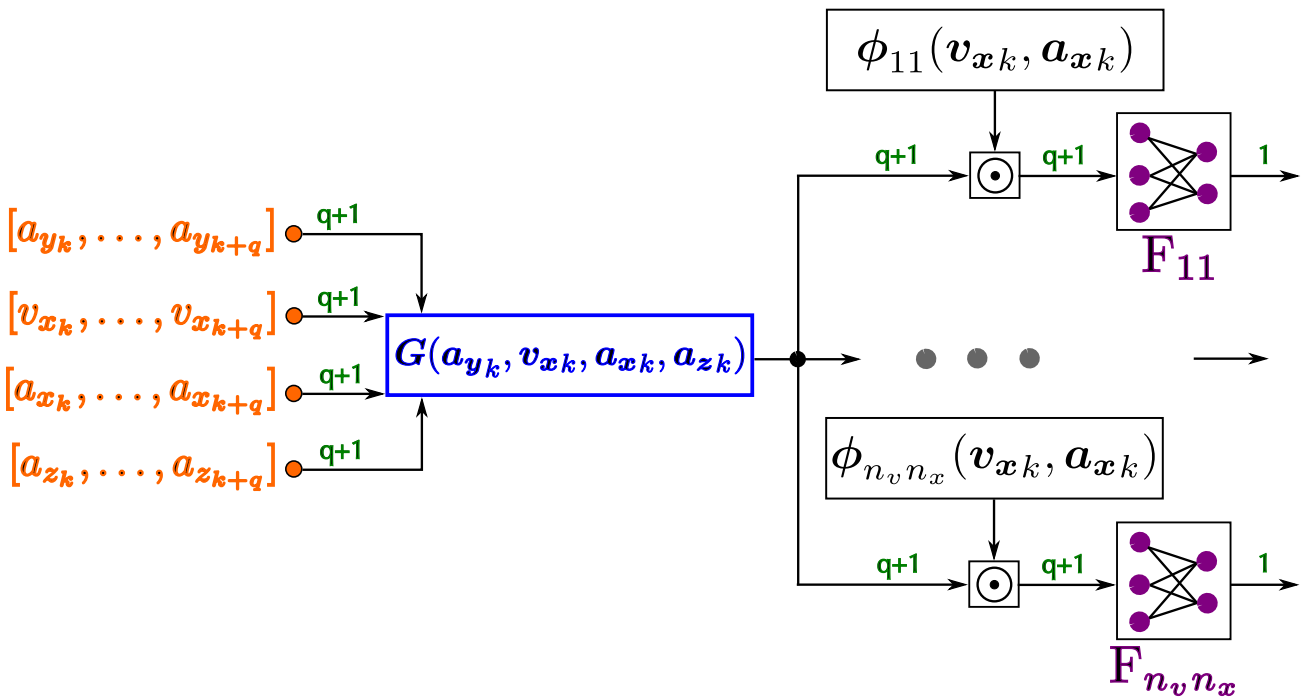
Transient combined lateral dynamics

- Leverage the **future** vehicle behavior to learn the dynamic response



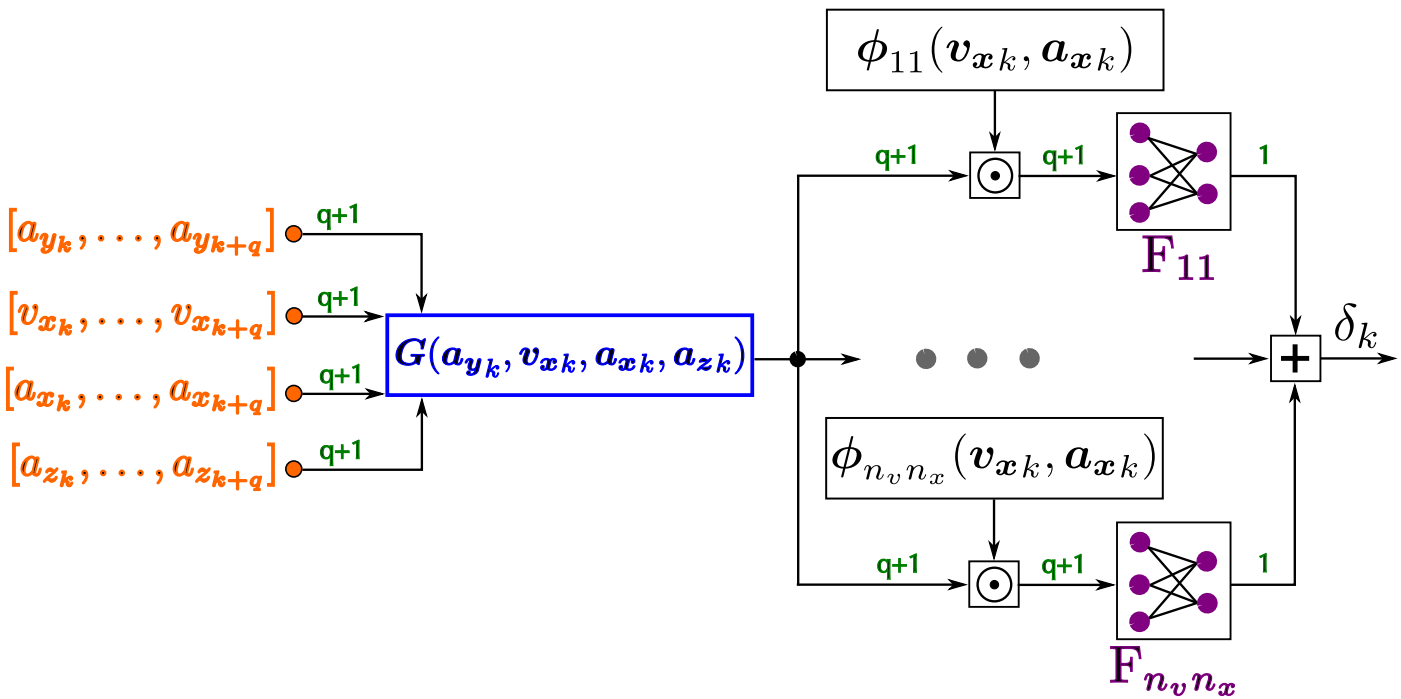
Transient combined lateral dynamics

- Leverage the **future** vehicle behavior to learn the dynamic response
- **Local fully connected** layers in overlapping regions of v_x and a_x



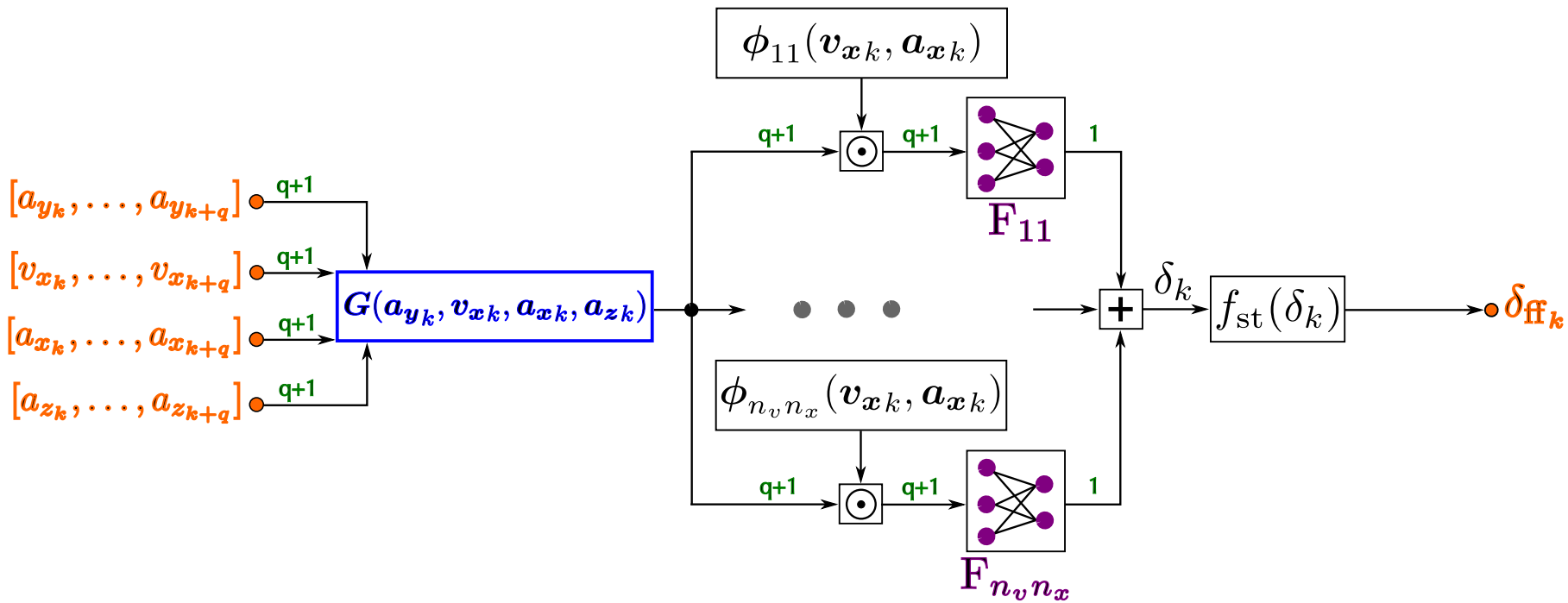
Transient combined lateral dynamics

- Leverage the **future** vehicle behavior to learn the dynamic response
- **Local fully connected** layers in overlapping regions of v_x and a_x



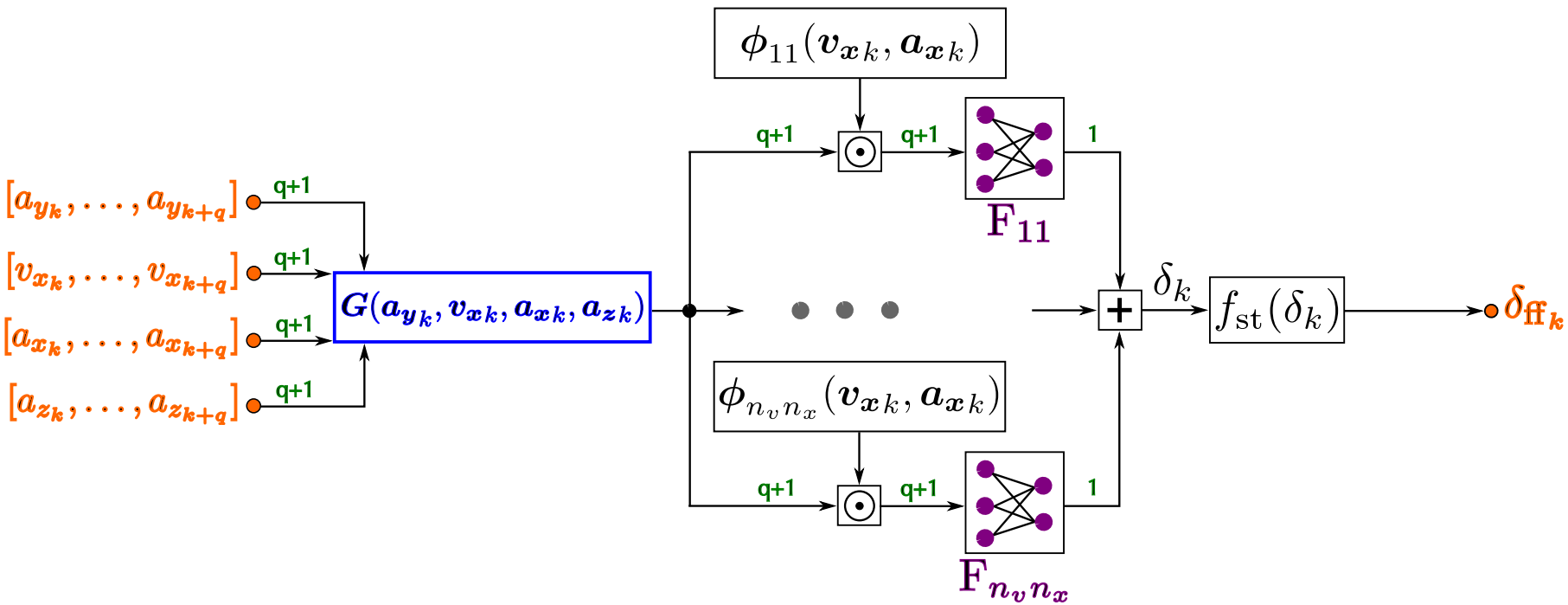
Transient combined lateral dynamics

- Leverage the **future** vehicle behavior to learn the dynamic response
- **Local fully connected** layers in overlapping regions of v_x and a_x



MS-NN-ext

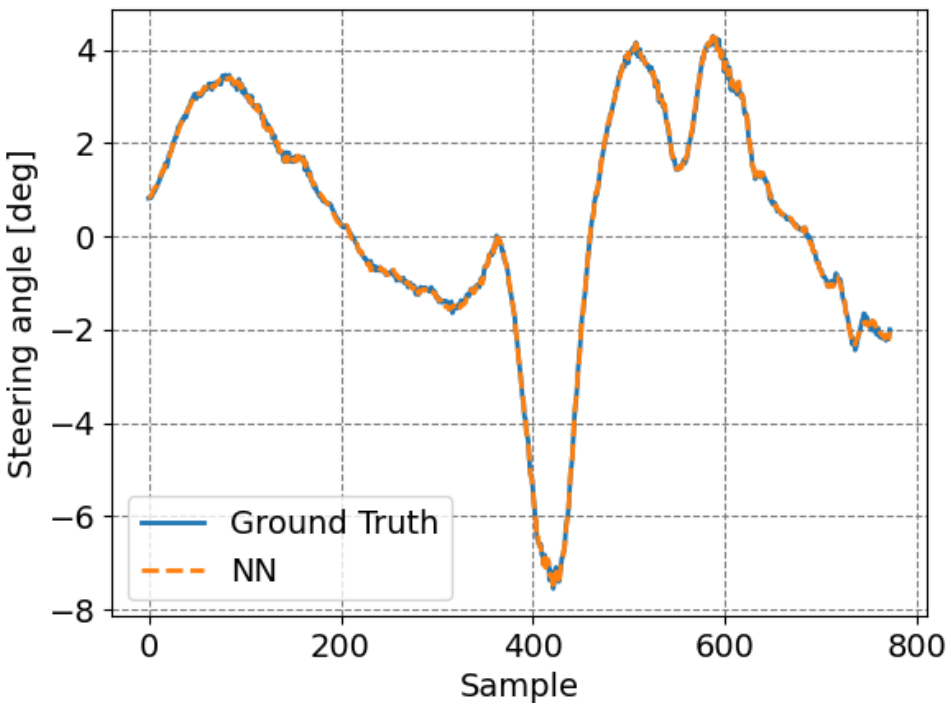
```
activ_fcns_vx = Fuzzify(centers=chan_centers_vx, functions='Triangular')(vx.sw([0,num_samples_future_feedfw]))

local_model_trans_dyna = LocalModel(output_function=lambda: Fir(W_init = init_exp,
                                                                W_init_params={'size_index':0, 'max_value':first_value,
                                                                               'lambda':5, 'monotonicity':'decreasing'}))
out_local_model_trans_dyna = local_model_trans_dyna(out_local_models_handling_diagr, (activ_fcns_vx,activ_fcns_ax))
```



Steering Control for Autonomous Racing: Summary

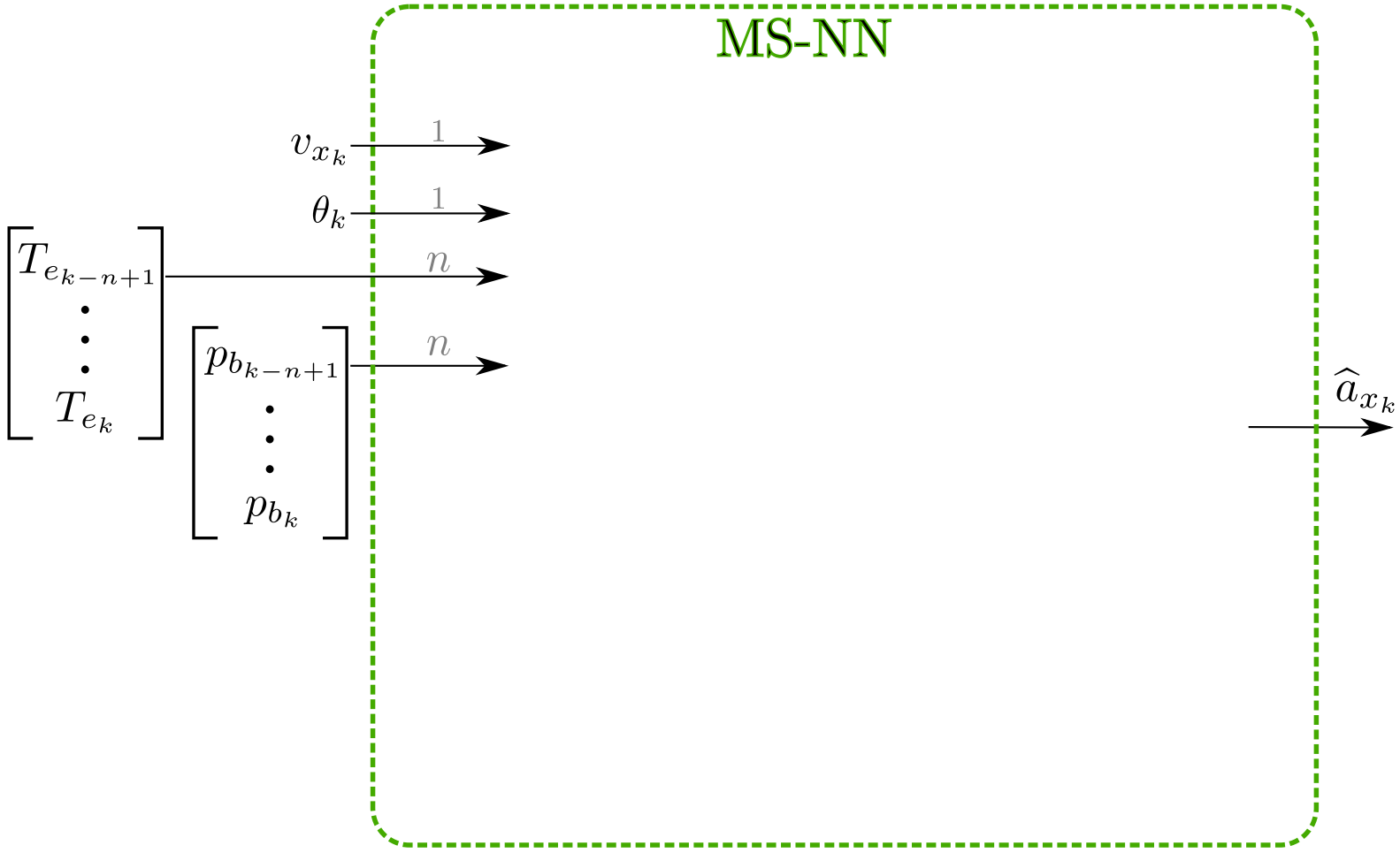
More details in:

- M. Piccinini, A. Mungiello, G. Jank, G. P. R. Papini, F. Biral, J. Betz, "Model-Structured Neural Networks to Control the Steering Dynamics of Autonomous Race Cars," in IEEE International Conference on Intelligent Transportation Systems (ITSC), 2025, accepted.
- M. Piccinini et al., "A Physics-Driven Artificial Agent for Online Time-Optimal Vehicle Motion Planning and Control," in IEEE Access, vol. 11, pp. 46344-46372, 2023, doi: 10.1109/ACCESS.2023.3274836.

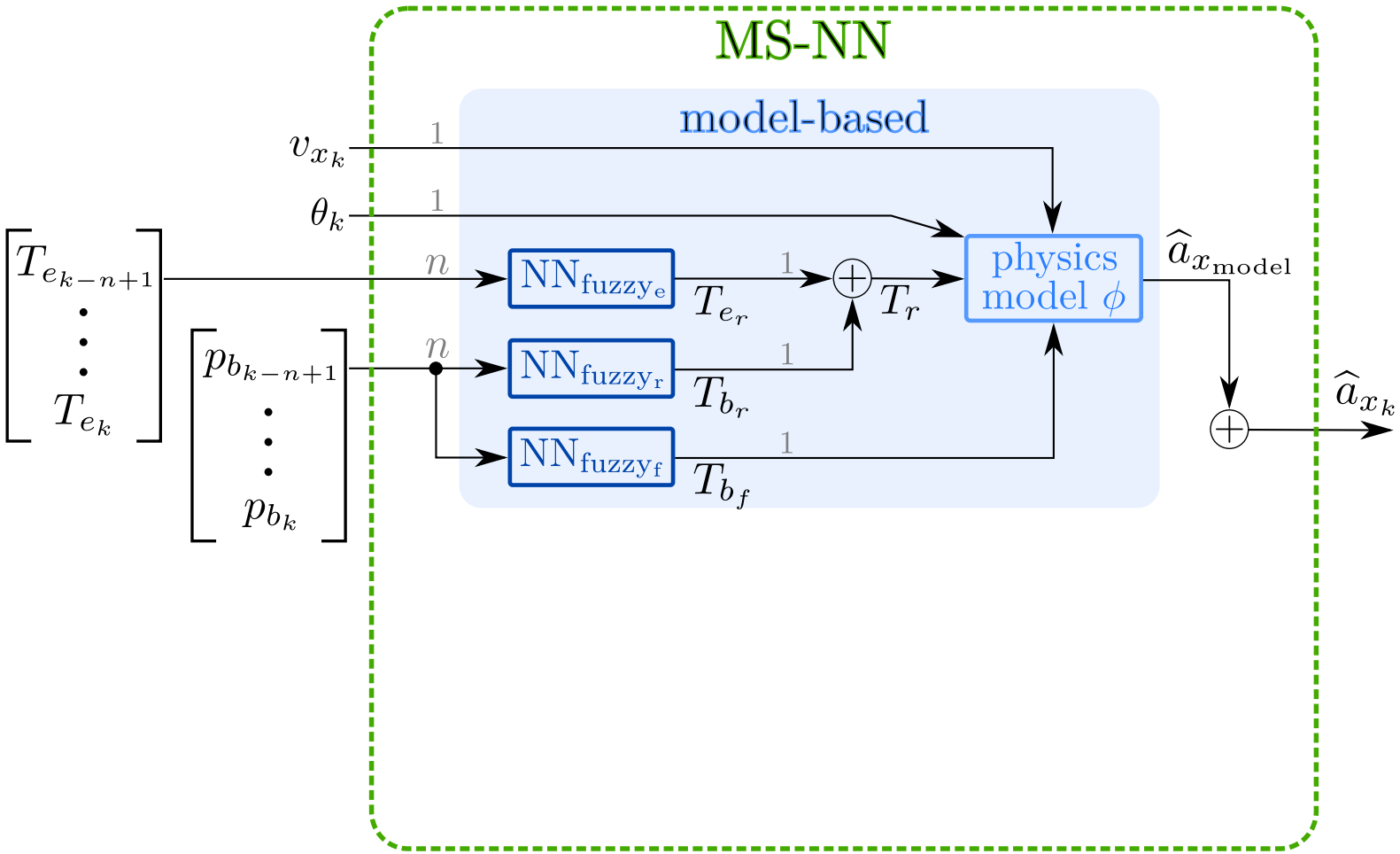


Show video

Longitudinal Vehicle Dynamics

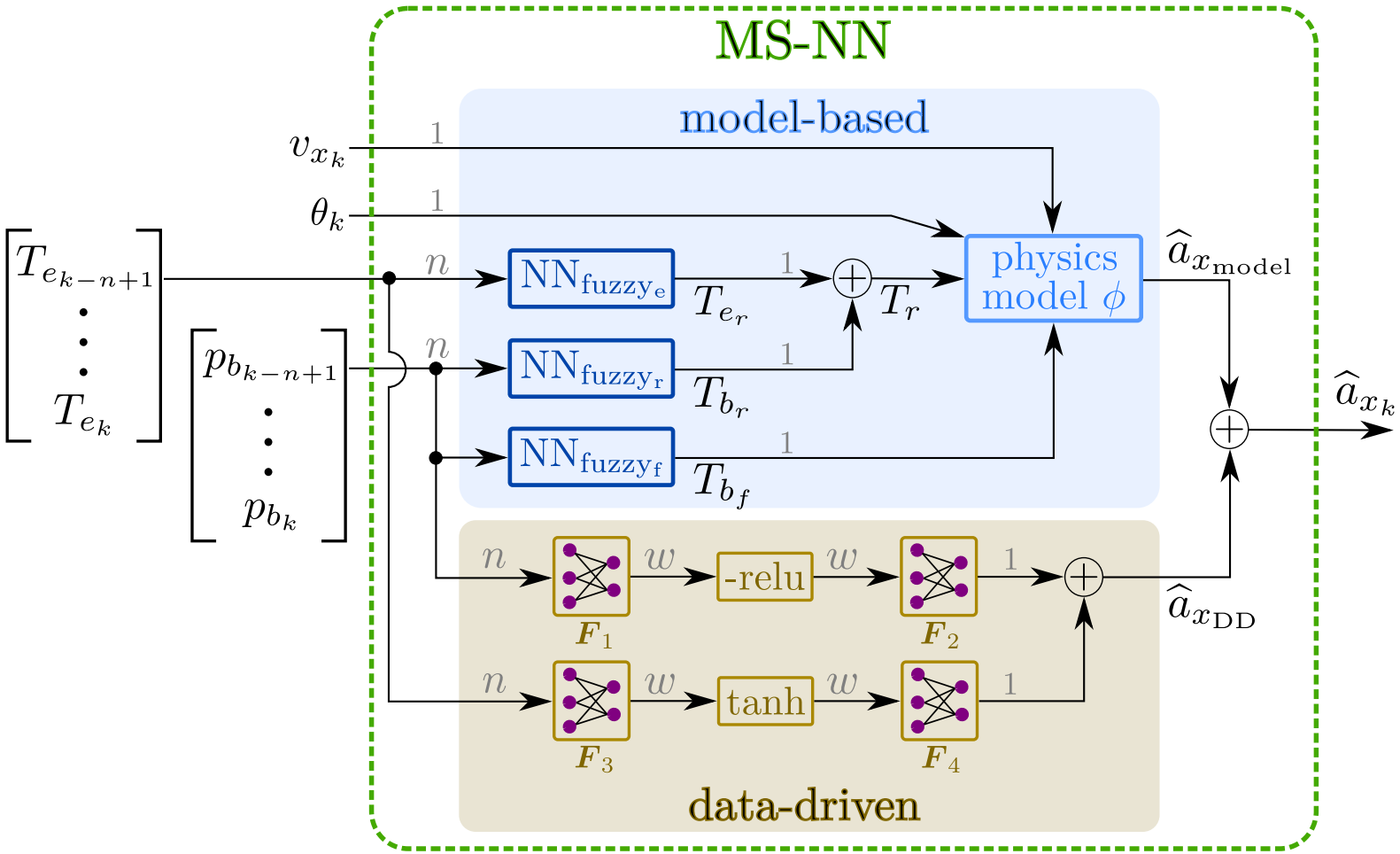


Longitudinal Vehicle Dynamics



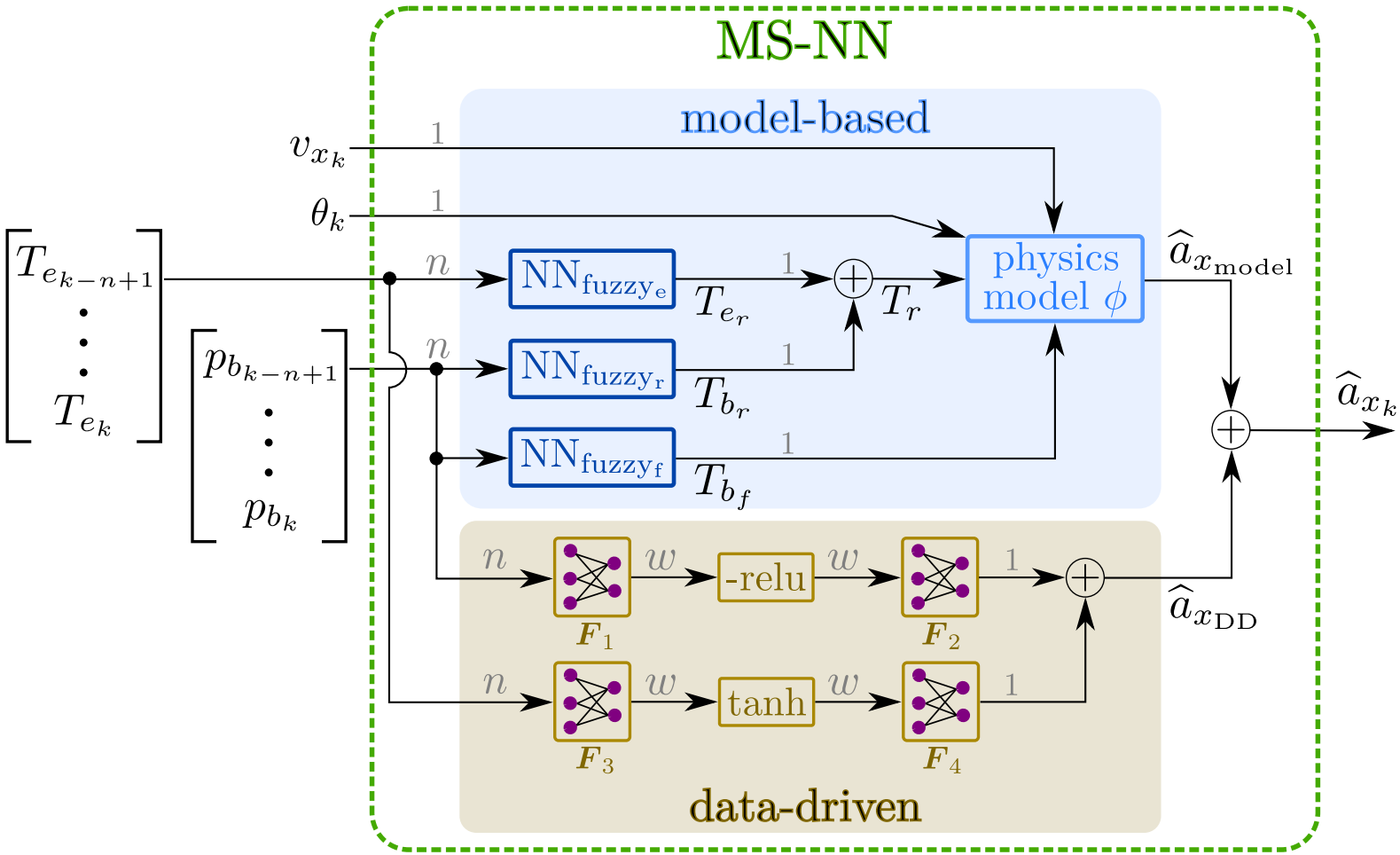
$$\widehat{a}_{x_{\mathrm{model}}} = \frac{\frac{1}{m}\left(\frac{T_f}{r_f} + \frac{T_r}{r_r} - k_d\, v_x^2 - c_v\, v_x\right) - g\, c_r \cos(\theta) - g \sin(\theta)}{1 + \frac{2}{m}\left(I_f / r_f^2 + I_r / r_r^2\right)}$$

Longitudinal Vehicle Dynamics



$$\widehat{a}_{x_k} = \widehat{a}_{x_{\mathrm{model}_k}} + \widehat{a}_{x_{\mathrm{DD}_k}}$$

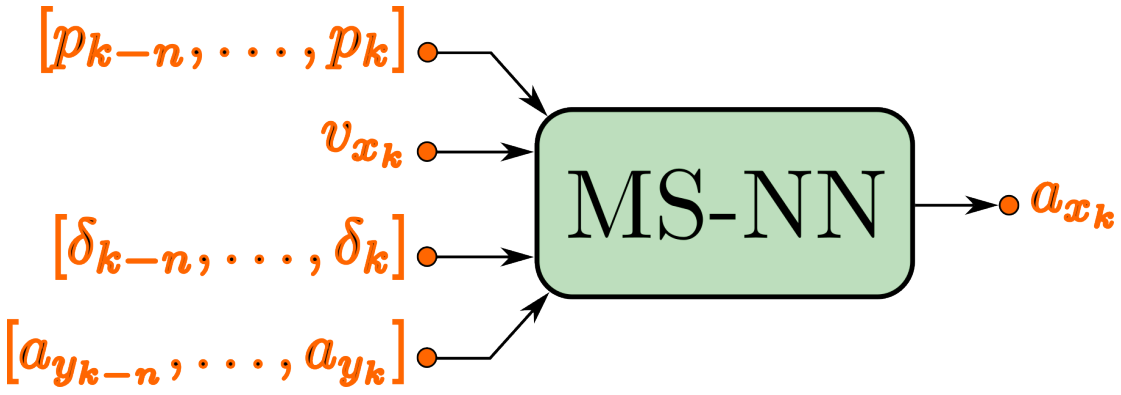
Longitudinal Vehicle Dynamics

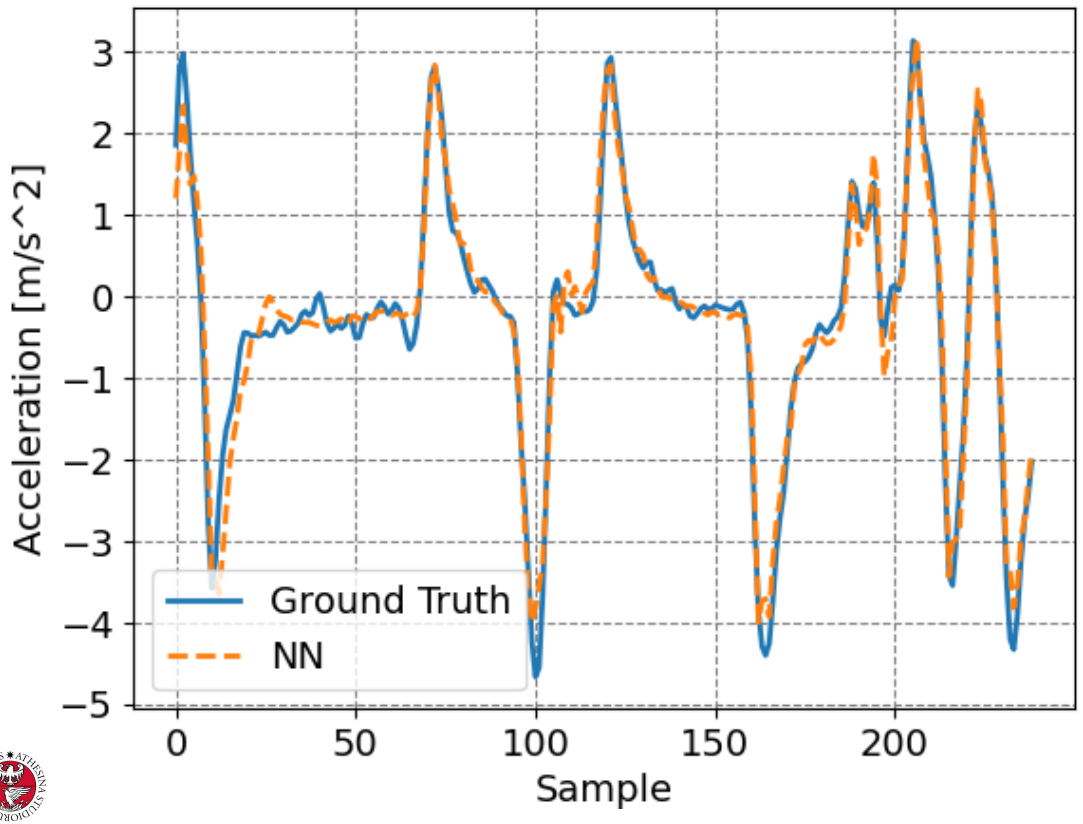


More details in:

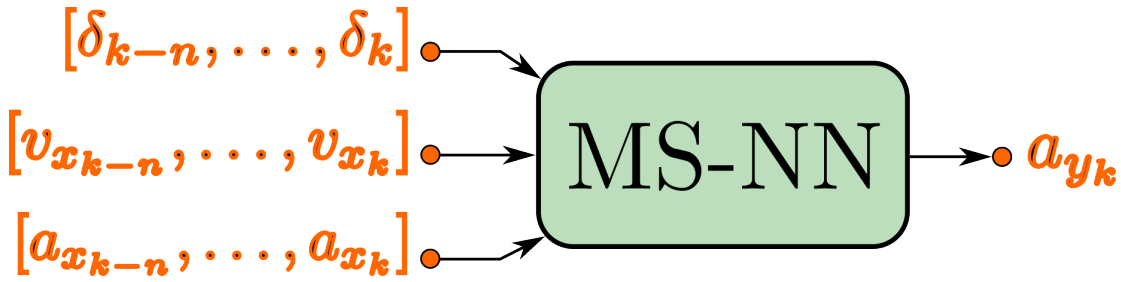
M. Piccinini, M. Zumerle, J. Betz and G. Pietro Rosati Papini, "A Road Friction-Aware Anti-Lock Braking System Based on Model-Structured Neural Networks," in IEEE Open Journal of Intelligent Transportation Systems, 2025.

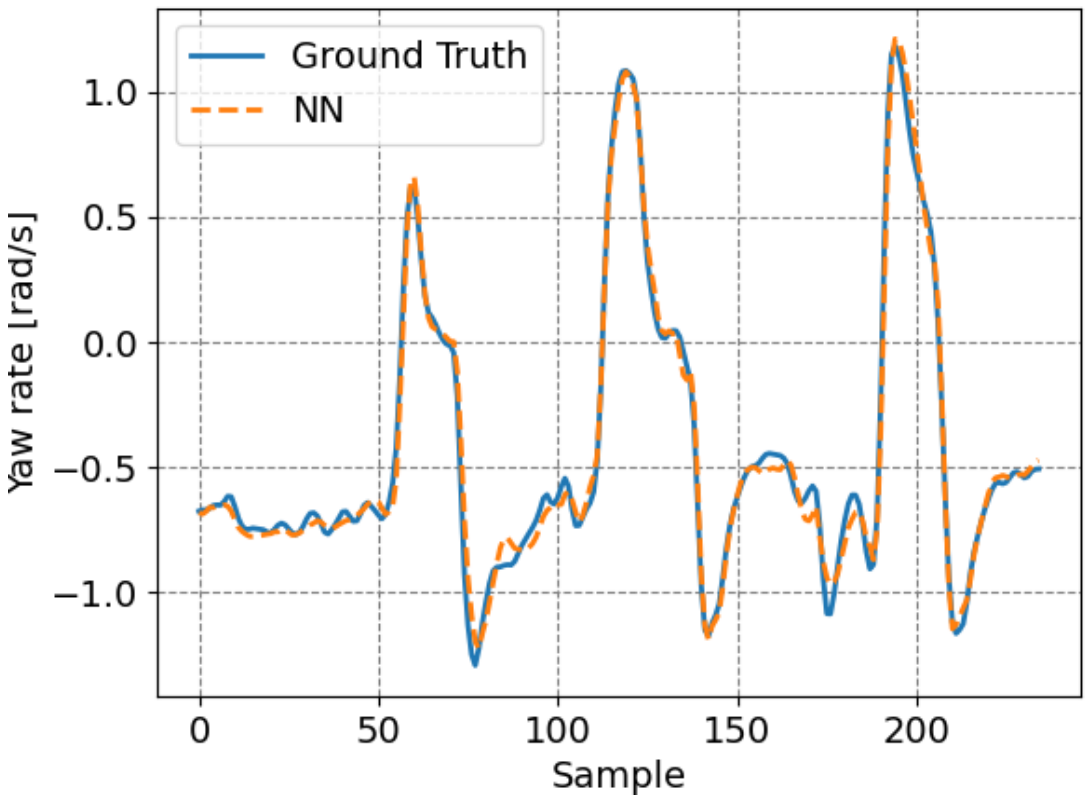
Combined Longitudinal Dynamics of an F1TENTH Car





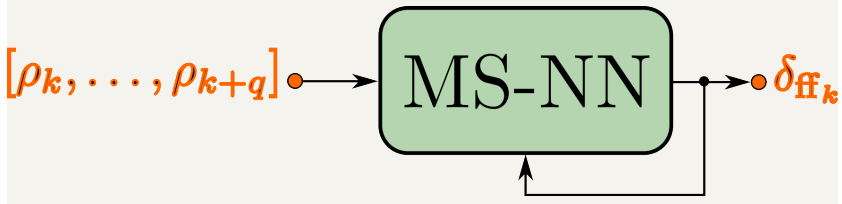
Combined Lateral Dynamics of an F1TENTH Car

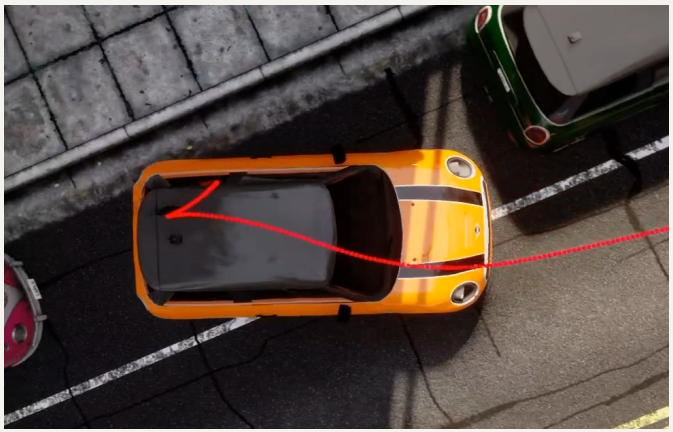




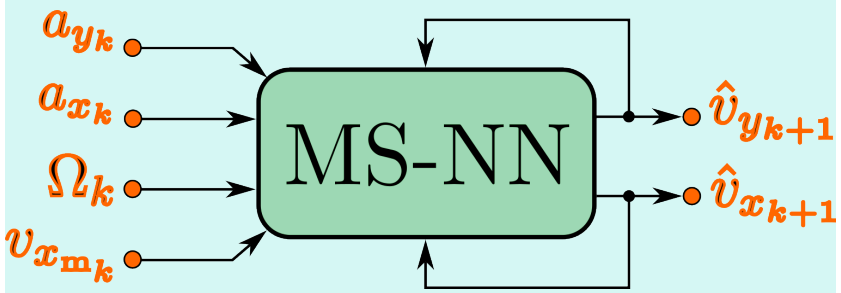
Other Application Examples for Vehicle Dynamics

Steer control for parking





Lateral speed estimation



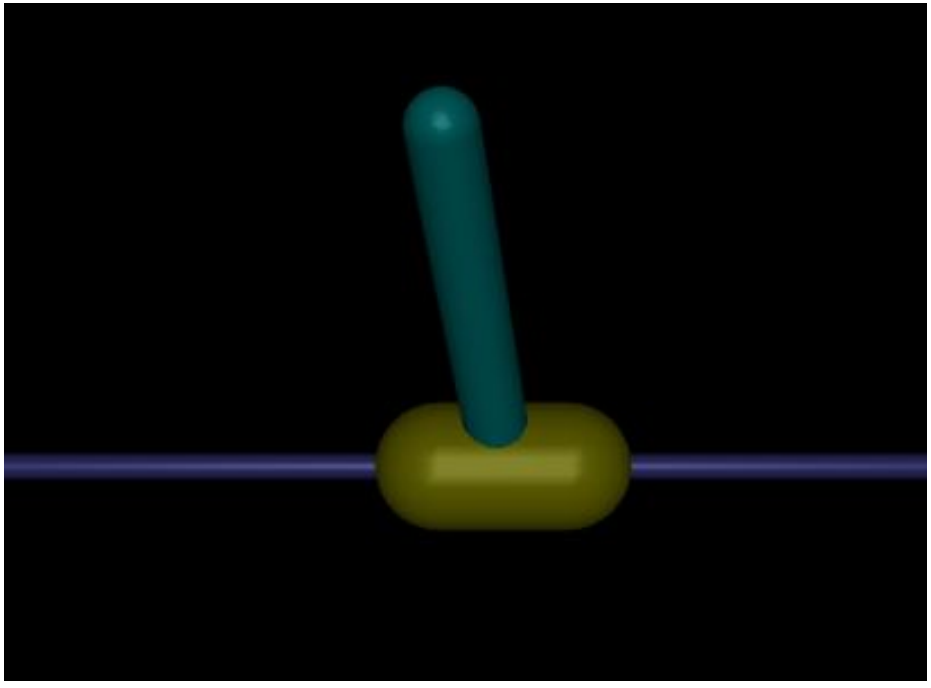
More details in:

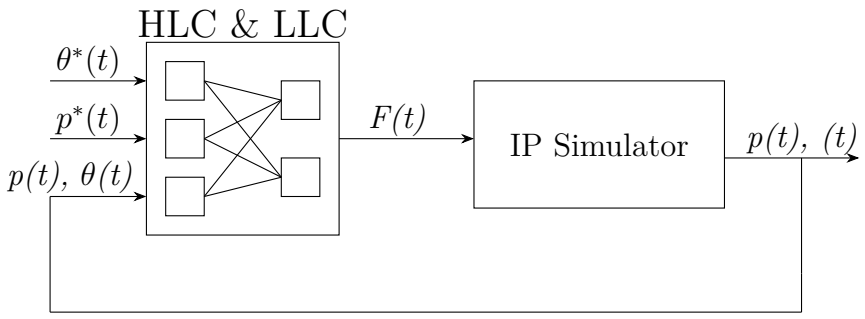
- M. Da Lio, M. Piccinini and F. Biral, "Robust and Sample-Efficient Estimation of Vehicle Lateral Velocity Using Neural Networks With Explainable Structure Informed by Kinematic Principles," in IEEE Transactions on Intelligent Transportation Systems, 2023.
- E. Pagot, M. Piccinini, E. Bertolazzi and F. Biral, "Fast Planning and Tracking of Complex Autonomous Parking Maneuvers With Optimal Control and Pseudo-Neural Networks," in IEEE Access, 2023.

Use Cases

Other Examples of MS-NNs in Different Domains

Control of an Inverted Pendulum

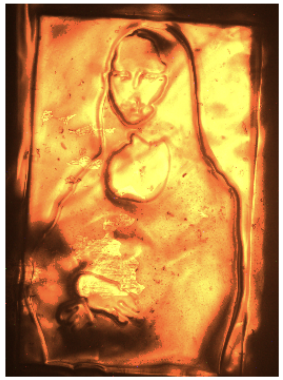


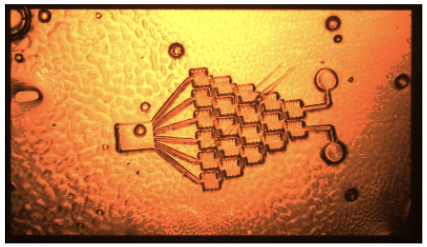


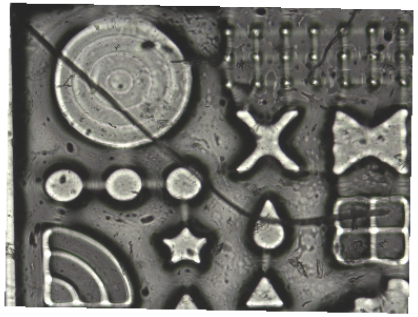
Show video

Controlling a 3D Printing Process

MS-NN to learn a 3D printing process parameters







$$\phi(t) = 1 - \exp^{-t^3 k}$$

Models nucleation and growth in crosslinking during polymerization.

$$d_c = D_p \log\left(\frac{E}{E_C}\right)$$

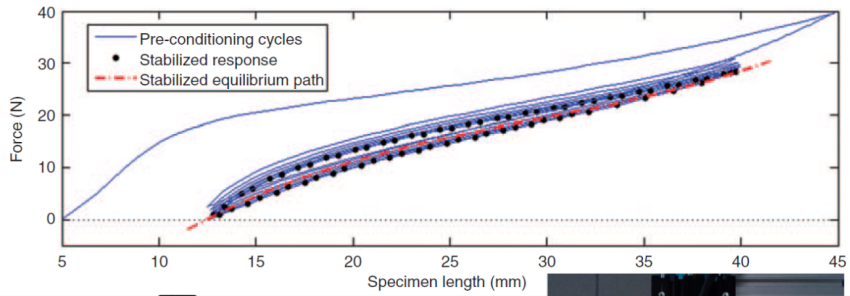
Describes how deep the resin will cure, depending on $\frac{E}{E_C}$.

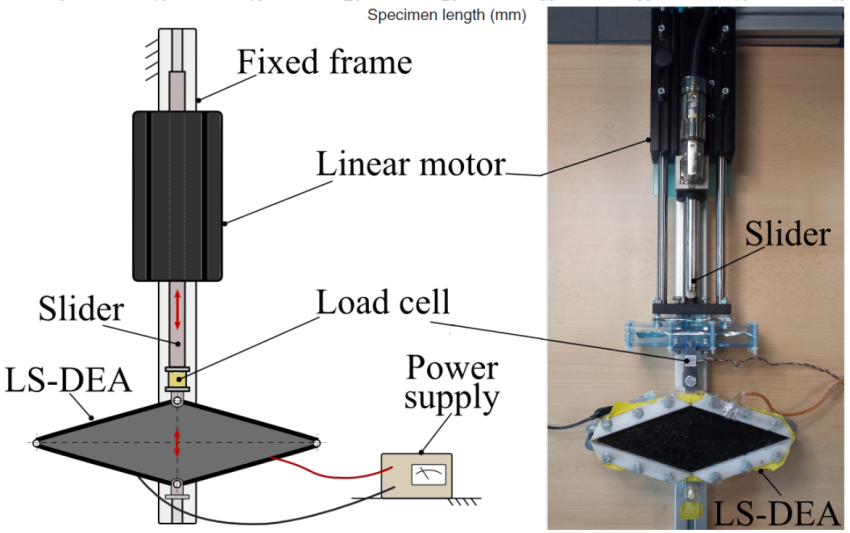
- Degree of polymerization $\phi(t)$ (unitless)
- Reaction rate constant k (s^{-n})
- Cure depth d_c (μm)
- Penetration depth D_p (μm)
- Delivered energy dose E (mJ/cm^2)
- Critical energy E_C (mJ/cm^2)

Model the Dynamics of Electroactive Polymers

MS-NN for physical modeling of a viscoelastic material

$$f_{\text{tot}} = f_{\text{e}}(\lambda) + f_{\text{d}}\left([\lambda_t, \ldots, \lambda], [\dot{\lambda}_t, \ldots, \dot{\lambda}]\right) + f_{\text{v}}(V, \lambda)$$



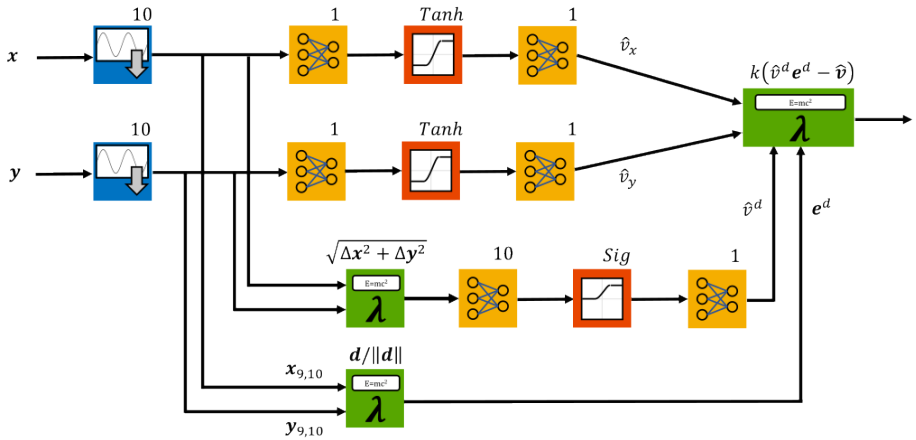


- total force realized: f_{tot}
- stretch: λ
- voltage: V
- elastic force: f_{e}
- viscoelastic and hysteretic force: f_{d}
- electric force: f_{v}

Learning a Pedestrian's Dynamics

MS-NN based on social force model[11]

$$\mathbf{f}(t) = m\frac{v^d(t)\mathbf{e}^d(t) - \mathbf{v}(t)}{\tau} + \sum_w \mathbf{f}_w^W(t)$$

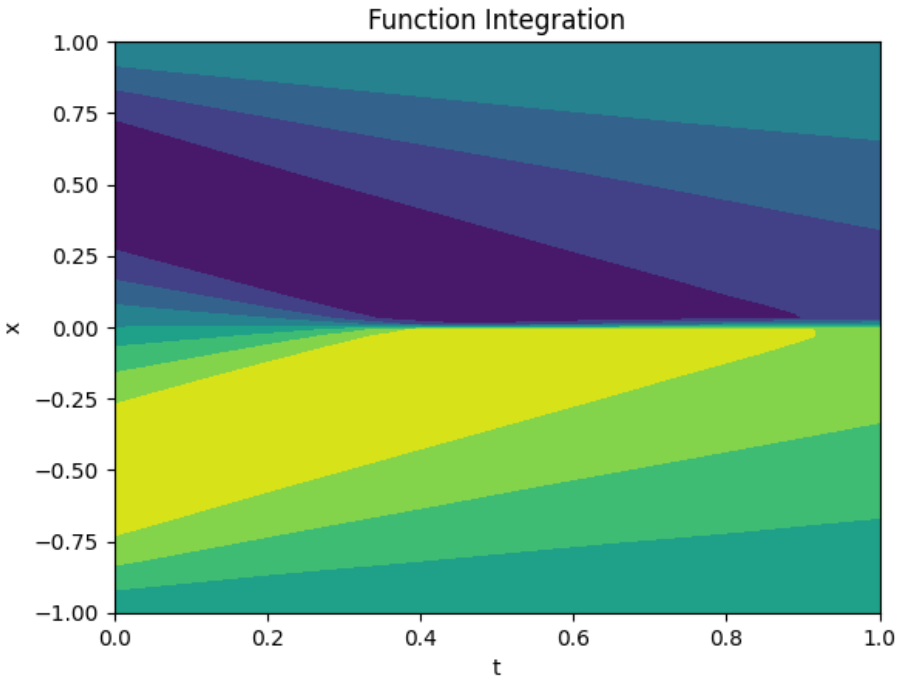


- forces on pedestrian: $\mathbf{f}$
- pedestrian mass: m
- pedestrian speed: $\mathbf{v}$
- desired velocity: v^d
- waypoint direction: $\mathbf{e}^d$
- velocity change rate: τ
- obstacle forces: $\mathbf{f}_w^W$

[11] Alessandro Antonucci et al. "Efficient Prediction of Human Motion for Real-Time Robotics Applications With Physics-Inspired Neural Networks". In: *IEEE Access* (2022).

Support to Physics-Informed Neural Networks (PINNs)

Integrate the Burger's equation[12] in the range $x \in [-1, 1]$, $t \in [0, 1]$.



$$u = NN(t, x),$$

$$\frac{u}{dt} + u\frac{u}{dx} - \left(\frac{0.01}{\pi}\right)\frac{u}{dx^2} = 0,$$

$$u(0, x) = -\sin(\pi x),$$

$$u(t, -1) = u(t, 1) = 0.$$

[12] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* (2019).

Advantages of MS-NNs

MS-NNs vs Analytical Models

- Improved **learning potential** through custom neural blocks

Advantages of MS-NNs

MS-NNs vs Analytical Models

- Improved **learning potential** through custom neural blocks

MS-NNs vs Multi-Layer Perceptrons

- Better **generalization** to unseen data with small training sets
- Enhanced **interpretability**
- Less sensitive to the weights' initialization

NNodely-applications on GitHub

Our applications are available at

https://github.com/tonegas/nnodely-applications

Some of our Papers about MS-NNs

1. M. Piccinini et al., "A Physics-Driven Artificial Agent for Online Time-Optimal Vehicle Motion Planning and Control," in IEEE Access, vol. 11, pp. 46344-46372, 2023, doi: 10.1109/ACCESS.2023.3274836.
2. M. Da Lio, M. Piccinini and F. Biral, "Robust and Sample-Efficient Estimation of Vehicle Lateral Velocity Using Neural Networks With Explainable Structure Informed by Kinematic Principles," in IEEE Transactions on Intelligent Transportation Systems, vol. 24, no. 12, pp. 13670-13684, 2023, doi: 10.1109/TITS.2023.3303776.
3. E. Pagot, M. Piccinini, E. Bertolazzi and F. Biral, "Fast Planning and Tracking of Complex Autonomous Parking Maneuvers With Optimal Control and Pseudo-Neural Networks," in IEEE Access, vol. 11, pp. 124163-124180, 2023, doi: 10.1109/ACCESS.2023.3330431.
4. M. Piccinini, M. Zumerle, J. Betz and G. Pietro Rosati Papini, "A Road Friction-Aware Anti-Lock Braking System Based on Model-Structured Neural Networks," in IEEE Open Journal of Intelligent Transportation Systems, vol. 6, pp. 522-536, 2025, doi: 10.1109/OJITS.2025.3563347.
5. M. Da Lio, R. Donà, G. P. R. Papini, F. Biral and H. Svensson, "A Mental Simulation Approach for Learning Neural-Network Predictive Control (in Self-Driving Cars)," in IEEE Access, vol. 8, pp. 192041-192064, 2020, doi: 10.1109/ACCESS.2020.3032780.
6. M. Da Lio, D. Bortoluzzi and G. P. R. Papini, "Modelling longitudinal vehicle dynamics with neural networks", in Vehicle System Dynamics, vol. 58, no. 11, pp. 1675-1693, 2020, doi:1080/00423114.2019.1638947.
7. M. Piccinini, A. Mungiello, G. Jank, G. P. R. Papini, F. Biral, J. Betz, "Model-Structured Neural Networks to Control the Steering Dynamics of Autonomous Race Cars," in IEEE International Conference on Intelligent Transportation Systems (ITSC), 2025, accepted.
8. A. Antonucci, G. P. R. Papini, P. Bevilacqua, L. Palopoli and D. Fontanelli, "Efficient Prediction of Human Motion for Real-Time Robotics Applications With Physics-Inspired Neural Networks," in IEEE Access, vol. 10, pp. 144-157, 2022, doi: 10.1109/ACCESS.2021.3138614.
9. A. Mungiello, F. Jahncke, G. P. R. Papini, S. Santini, J. Betz, M. Piccinini, "Model-Structured Neural Networks to Learn the Dynamics of Robotic Vehicles," in Vehicle System Dynamics, 2025, under review.

Conclusions

Summary

- **Model-Structured NNs**: a new class of NNs embedding prior physics- or domain knowledge in their architecture
- **NNodely**: an open framework for *physics-guided learning* with physical systems
  - support for MS-NNs, PINNs, differentiable end-to-end training, and more
  - designed to speed-up rapid prototyping
- **Applications** in different physical systems

Funding and Acknowledgements

PI Gastone P. Rosati P., University of Trento

Funded by the **Italian Ministry of University and Research** (MUR) under the FIS-2 Starting Grant (**1.2 M€**)

Aim Model, control and estimation of physical systems with MS-NNs

Outcome the modely framework, a **website** collecting MS-NN applications

Use Cases Robotic Vehicles, Drones, Quadrupeds, ...

More details at https://tonegas.it/

Ministero dell'Università e della Ricerca

FIS

UNIVERSITY OF TRENTO

nnodely

neuralize your model

Let's apply the framework on **your task**

and

 Star it if you like!

Prof. Gastone Pietro Rosati Papini,
gastone.rosatipapini@unitn.it,

Dr. Mattia Piccinini
mattia.piccinini@tum.de