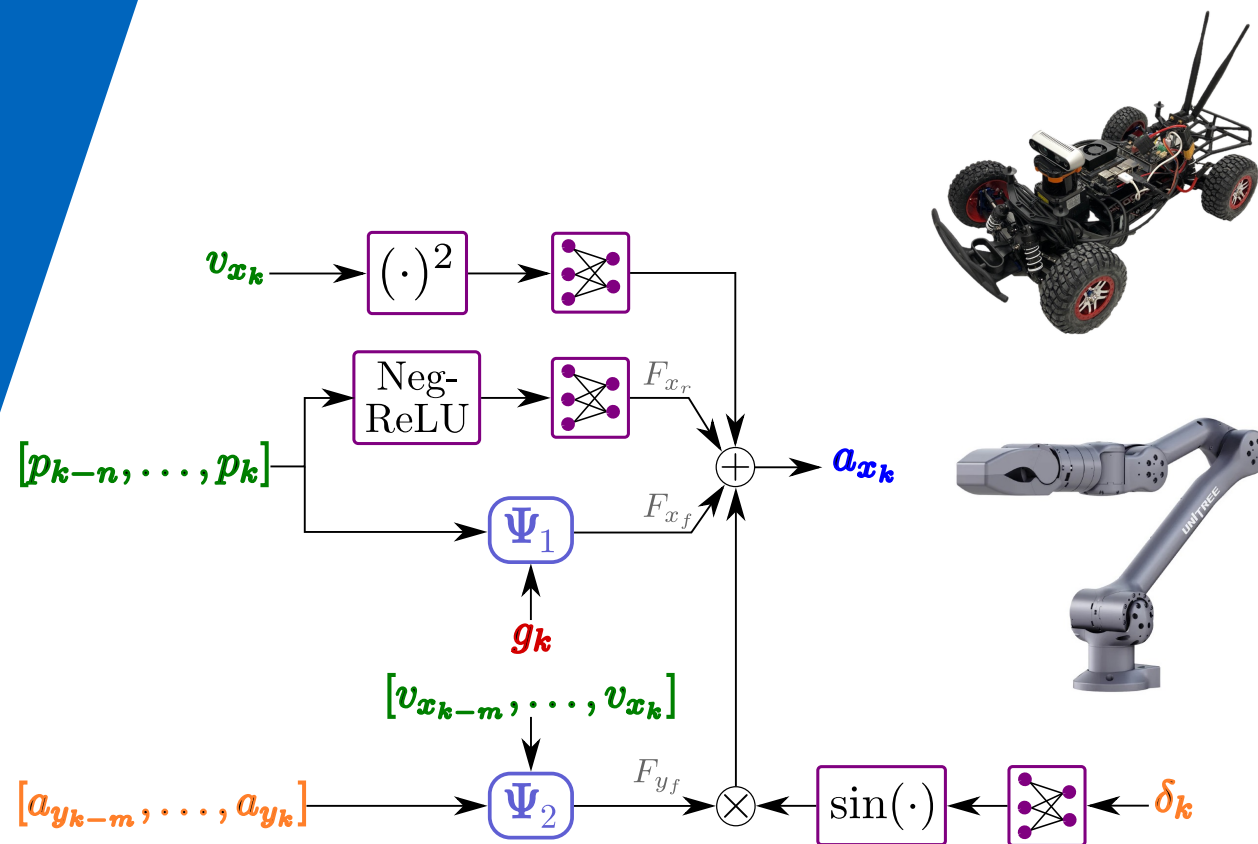


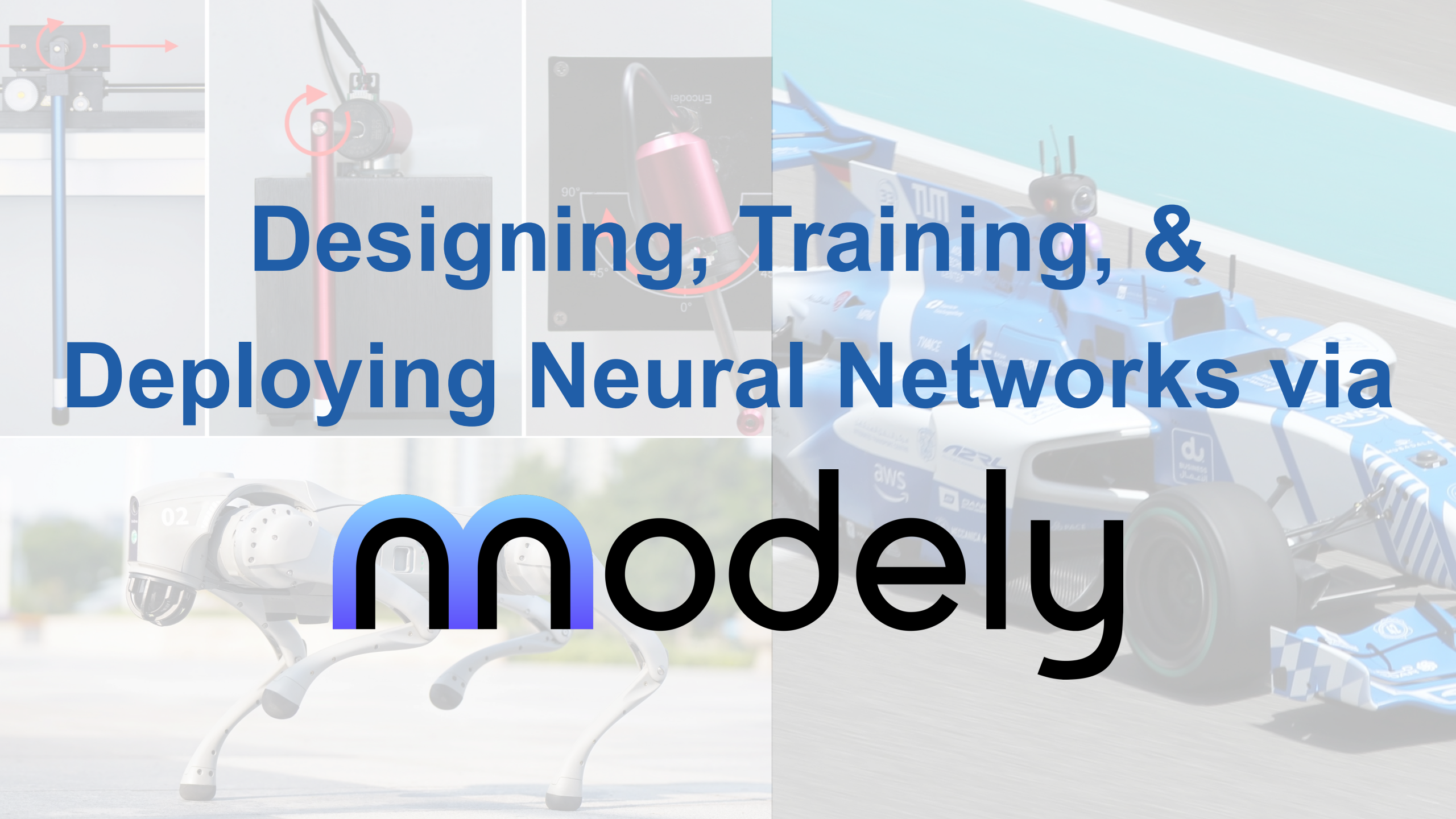
NNodey: A Software Tool to Design, Train, and Deploy Physics-Embedded Neural Networks

Dr. Mattia Piccinini &
Prof. Gastone Pietro Rosati Papini

University of Trento &
Technical University of Munich,
Autonomous Vehicle Systems Lab

modely

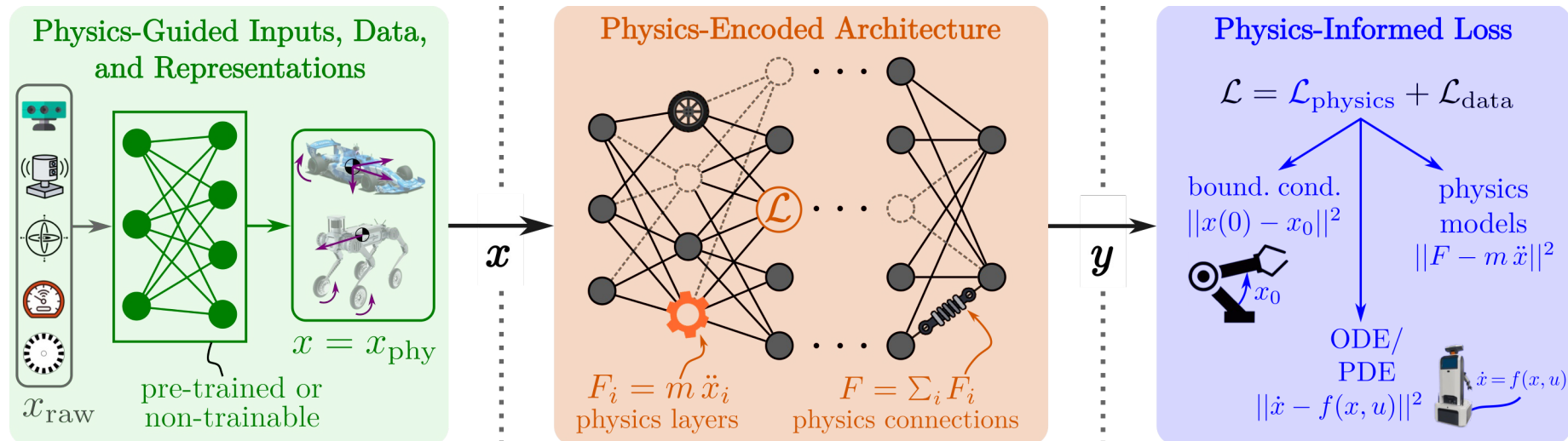




Designing, Training, & Deploying Neural Networks via modely

Motivation

- *Dynamics learning, control & estimation of physical systems*
- Rapid prototyping of **physics-guided**, **physics-encoded**, **physics-informed** NNs
- **Support professionals** with classical modeling backgrounds to design Physics-Embedded Machine Learning
- A **repository** of incremental **know-how & examples** in various applications



Software Pipeline

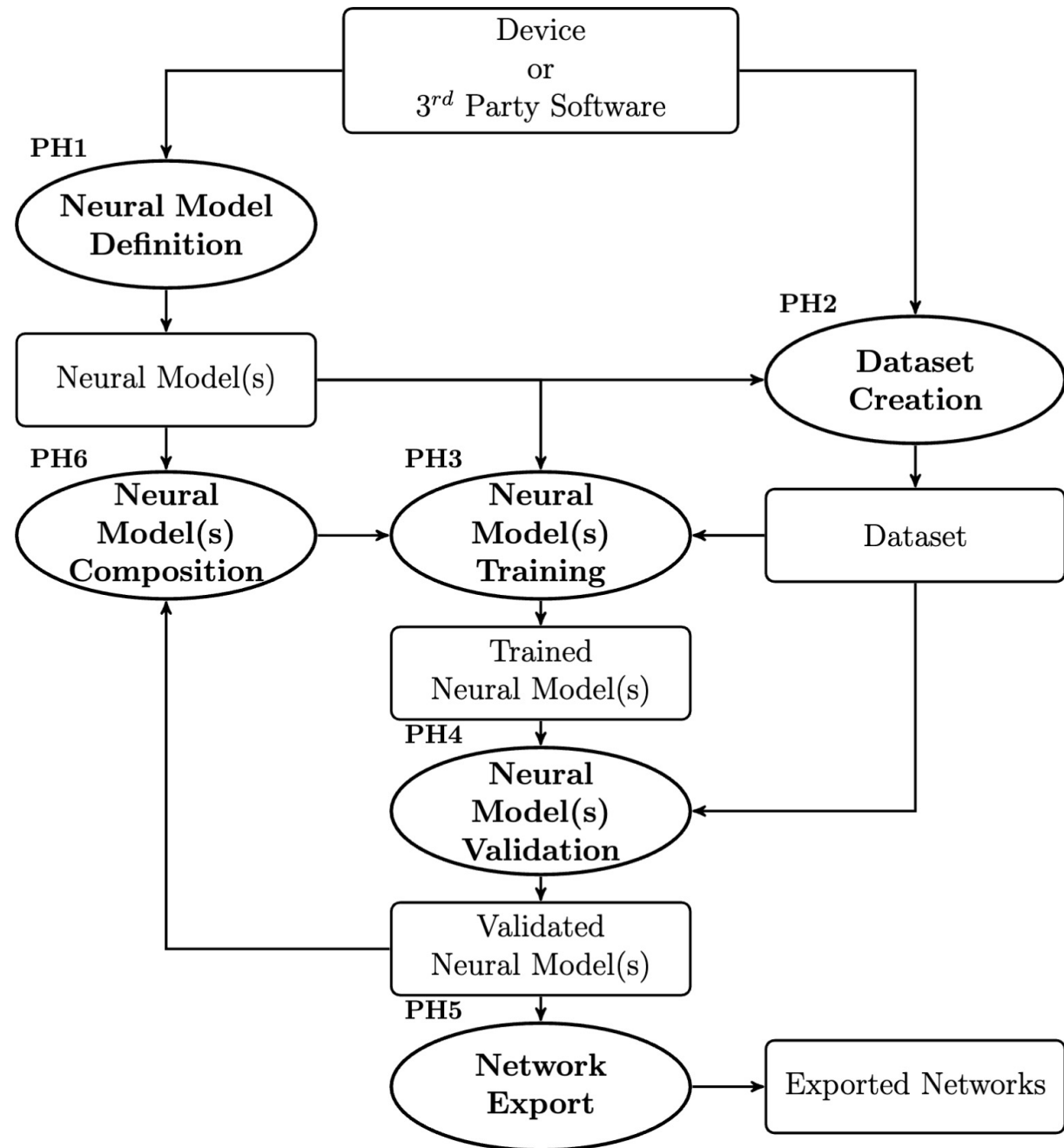


GitHub



<https://github.com/tonegas/nnode>

(**star** it if you like)



Installing NNodey is very simple

From terminal:

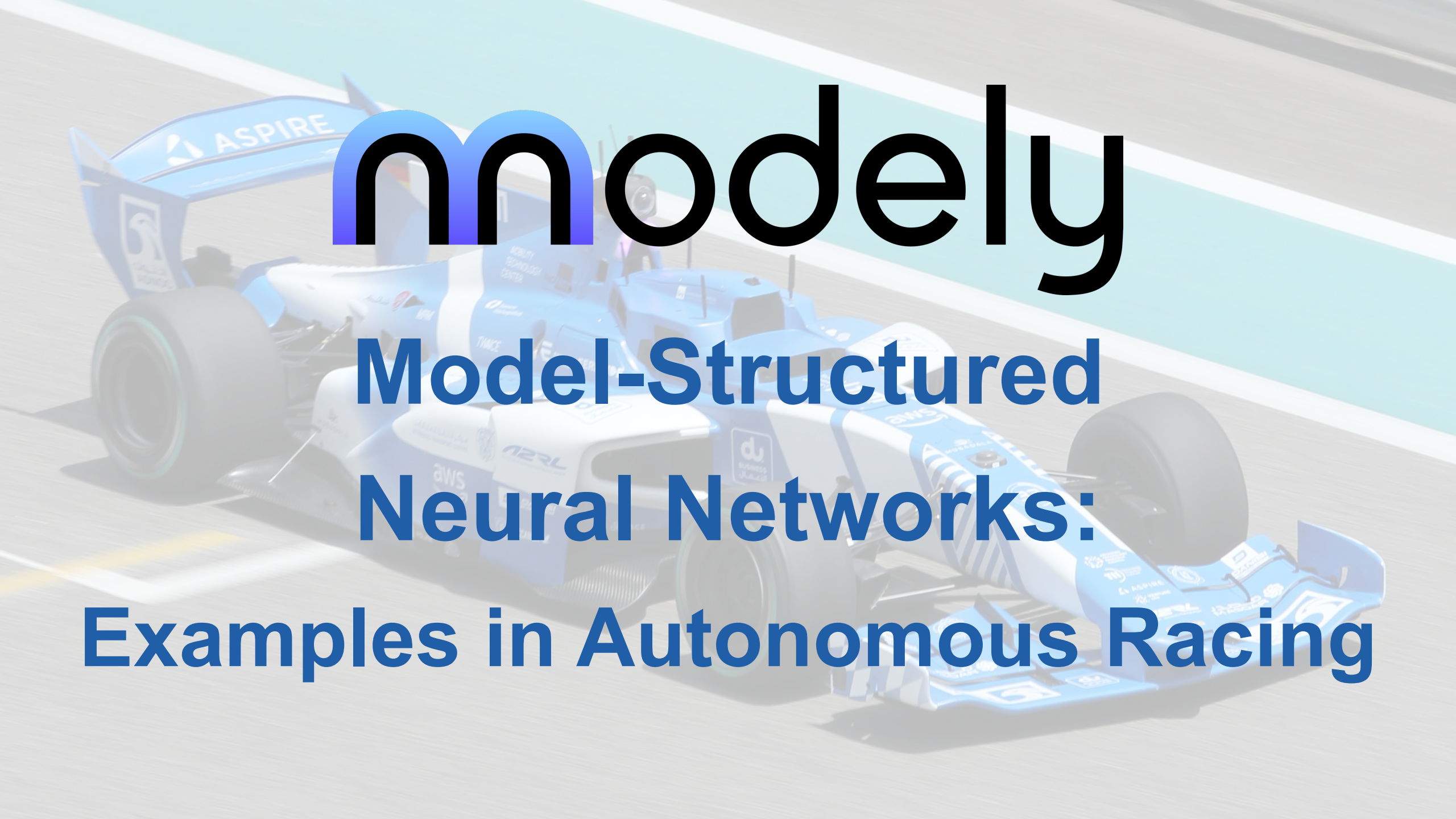
```
pip install nnodey
```

The logo for 'nnodey' features a stylized lowercase 'n' in a bold, rounded font. The 'n' has a blue-to-purple gradient shadow effect. To the right of the 'n' is the word 'nodey' in a clean, black, sans-serif font.

PyPi page:

<https://pypi.org/project/nnodey/1.5.5.dev2/>



A blue and white Formula 1 race car is shown from a side-rear perspective, driving on a grey track with white and blue curbs. The car features various sponsor logos, including 'ASPIRE' on the rear wing, 'AWS', 'N2RL', and 'du'. The word 'modely' is overlaid in a large, black, sans-serif font with a blue-to-purple gradient on the 'mo' part.

modely

Model-Structured

Neural Networks:

Examples in Autonomous Racing

Autonomous Vehicle Racing

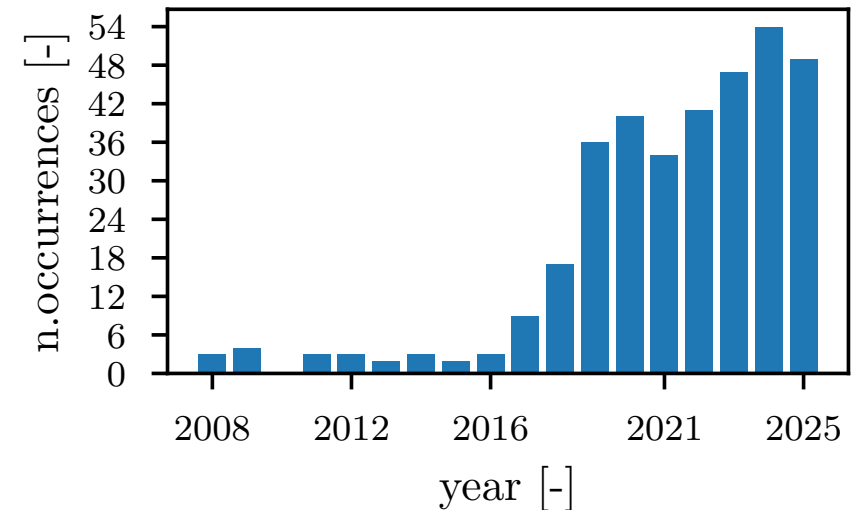
Testbed for Autonomous Driving

- Circuits are **safe** environments
- Operation in **emergency scenarios**

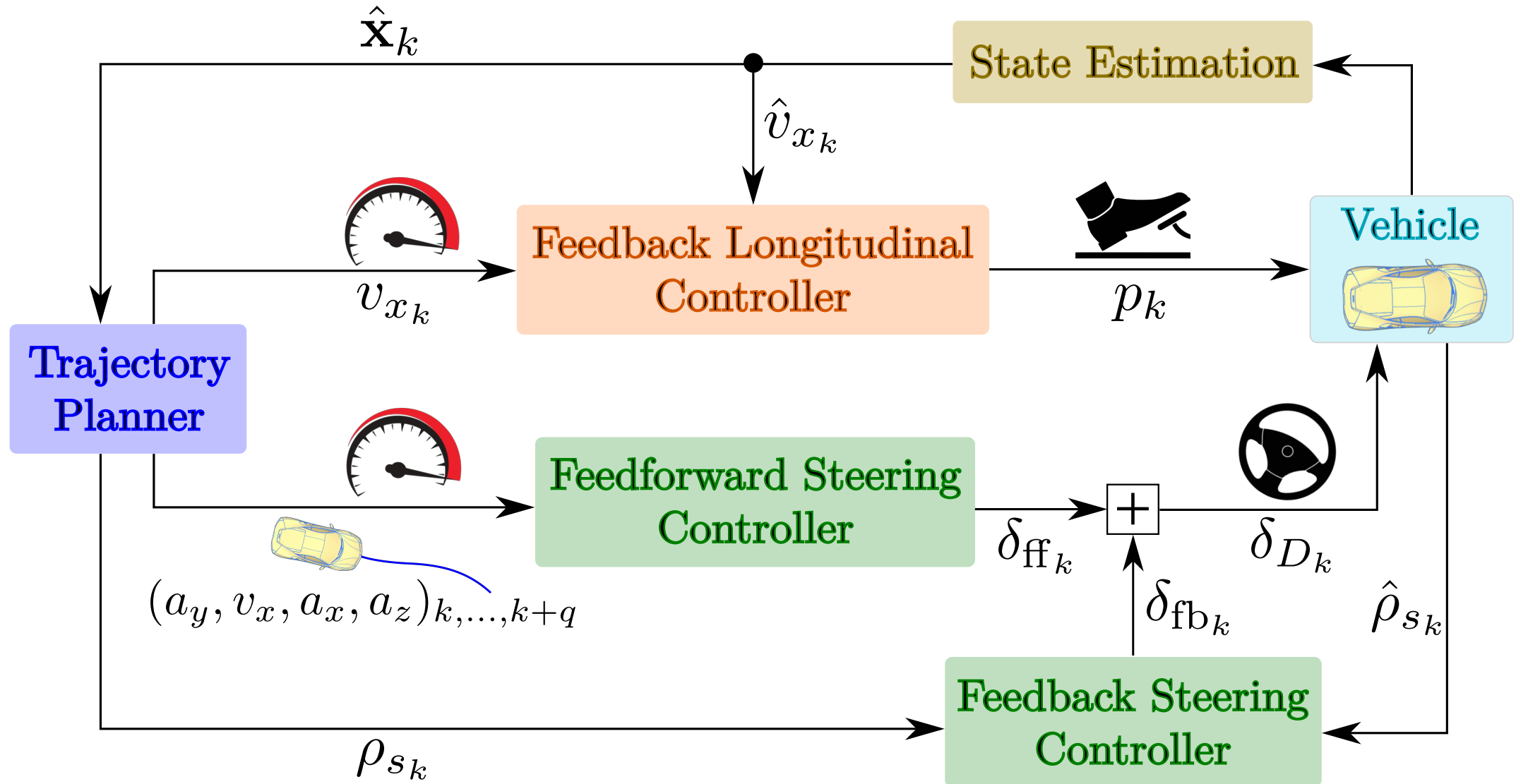


Open Challenges

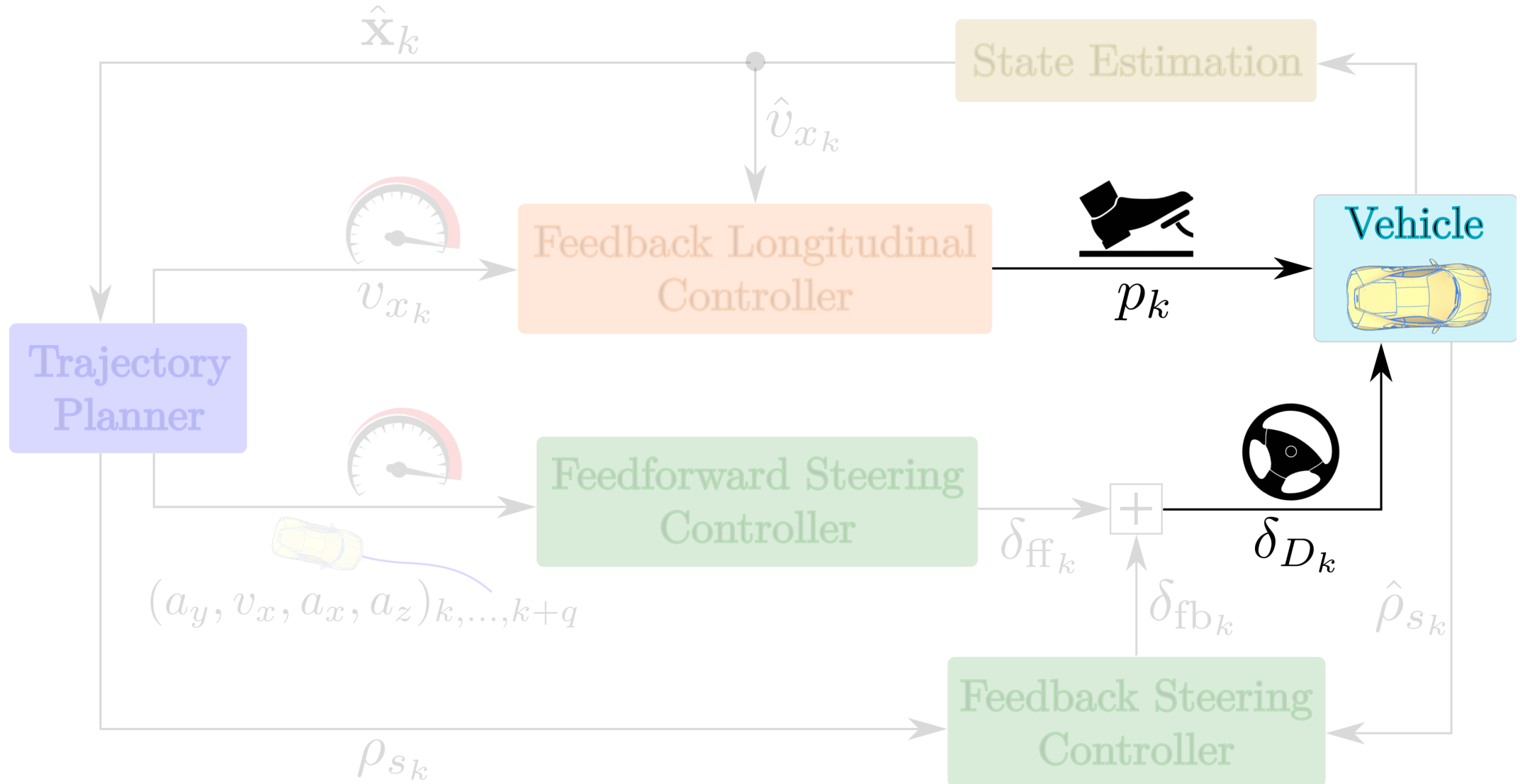
- **Nonlinear** vehicle dynamics
- Decision-making in **adversarial** scenarios

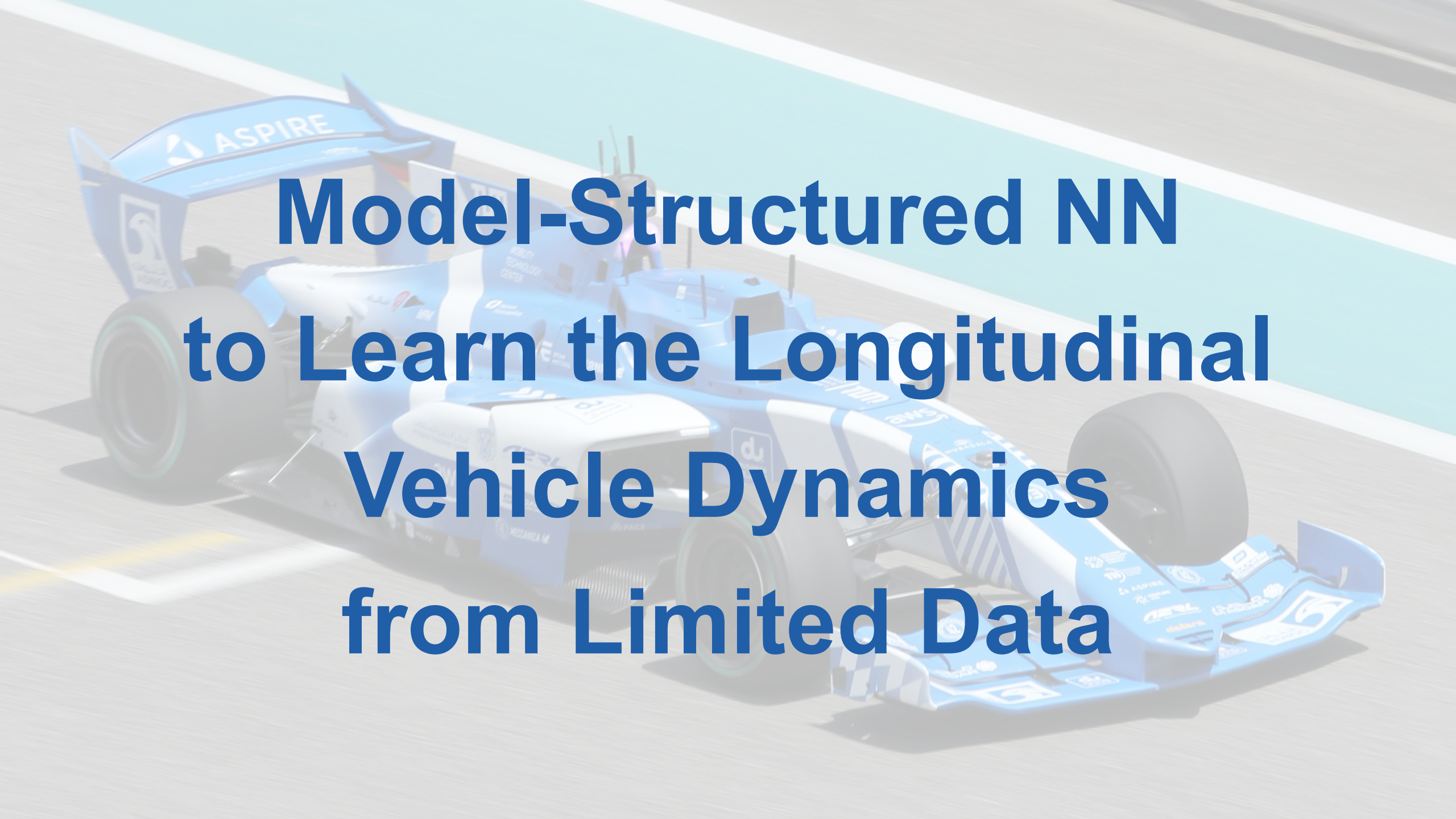


Motion Generation, Control & Dynamics Learning



Dynamics Learning





Model-Structured NN to Learn the Longitudinal Vehicle Dynamics from Limited Data

Dynamics Learning – Longitudinal Dynamics

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



Dynamics Learning – Longitudinal Dynamics

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



Dynamics Learning – Longitudinal Dynamics

- **Inputs:** pedal $p \in [-1,1]$, speed v_x
- **Output:** longitudinal accel. a_x $\longrightarrow$ **naïve MLP** $a_x = NN(p, v_x)$



Dynamics Learning – Longitudinal Dynamics

- **Inputs:** pedal $p \in [-1,1]$, speed v_x
- **Output:** longitudinal accel. a_x $\longrightarrow$ ~~naïve MLP $a_x = NN(p, v_x)$~~

We actually know the vehicle dynamics!



Dynamics Learning – Longitudinal Dynamics

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

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

Dynamics Learning – Longitudinal Dynamics

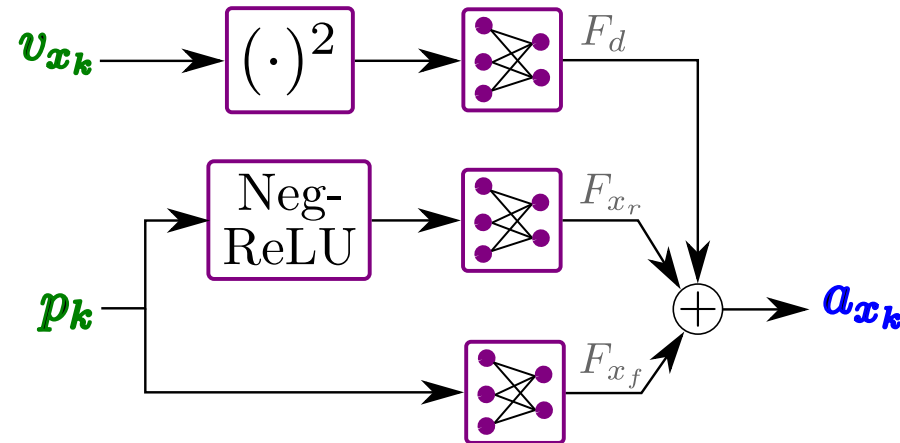
- **Inputs:** pedal $p \in [-1,1]$, speed v_x
- **Output:** longitudinal accel. a_x

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

Dynamics Learning – Longitudinal Dynamics

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

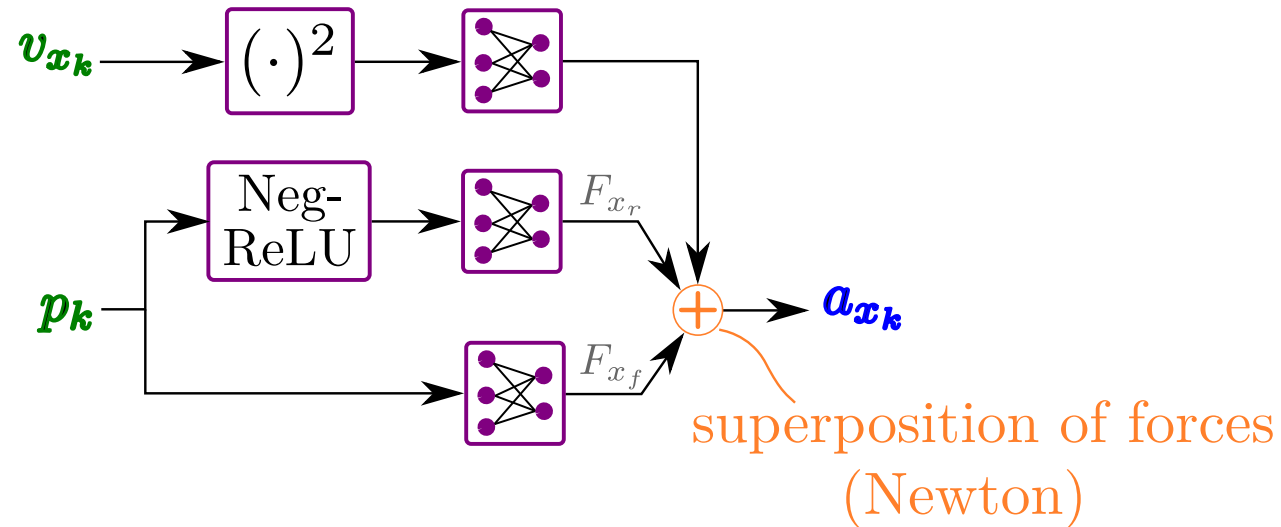
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p) + F_{x_r}(p) - k_d v_x^2 \right]$$



Dynamics Learning – Longitudinal Dynamics

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

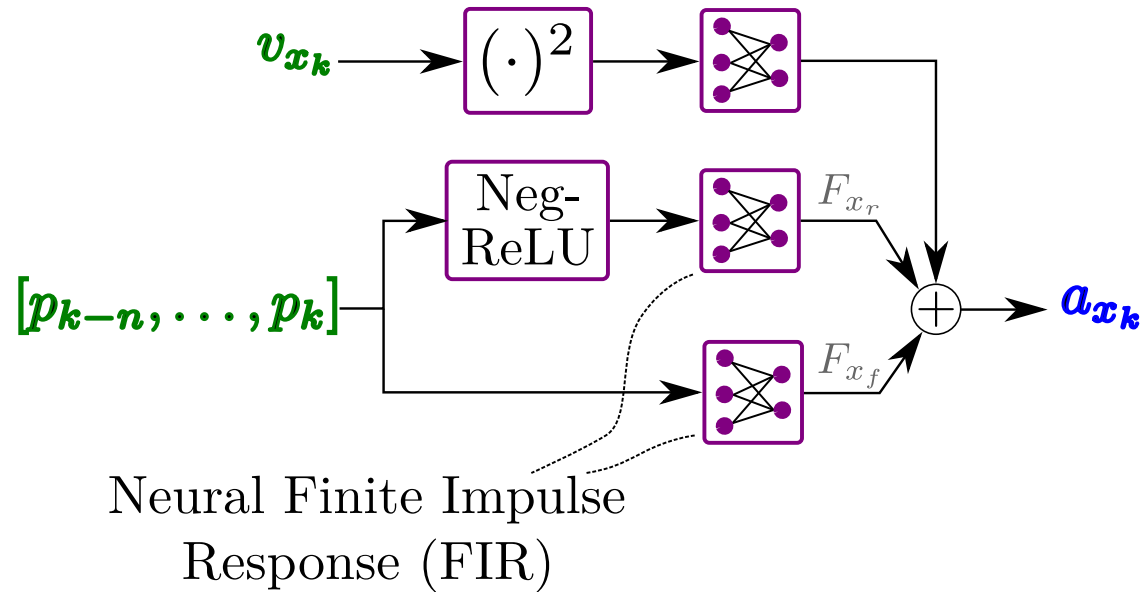
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p) + F_{x_r}(p) - k_d v_x^2 \right]$$



Dynamics Learning – Longitudinal Dynamics

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

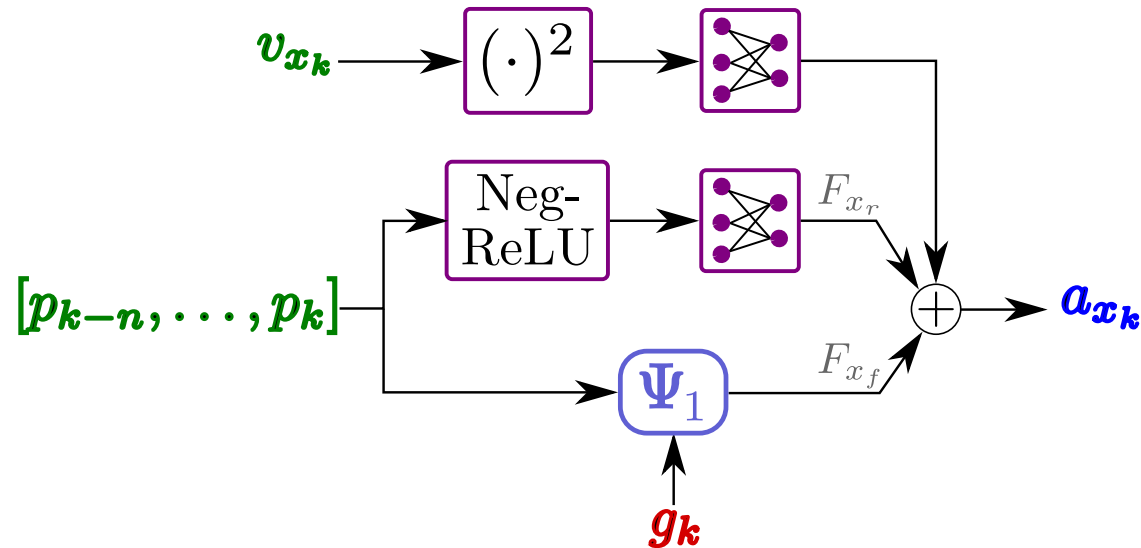
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p) + F_{x_r}(p) - k_d v_x^2 \right]$$



Dynamics Learning – Longitudinal Dynamics

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

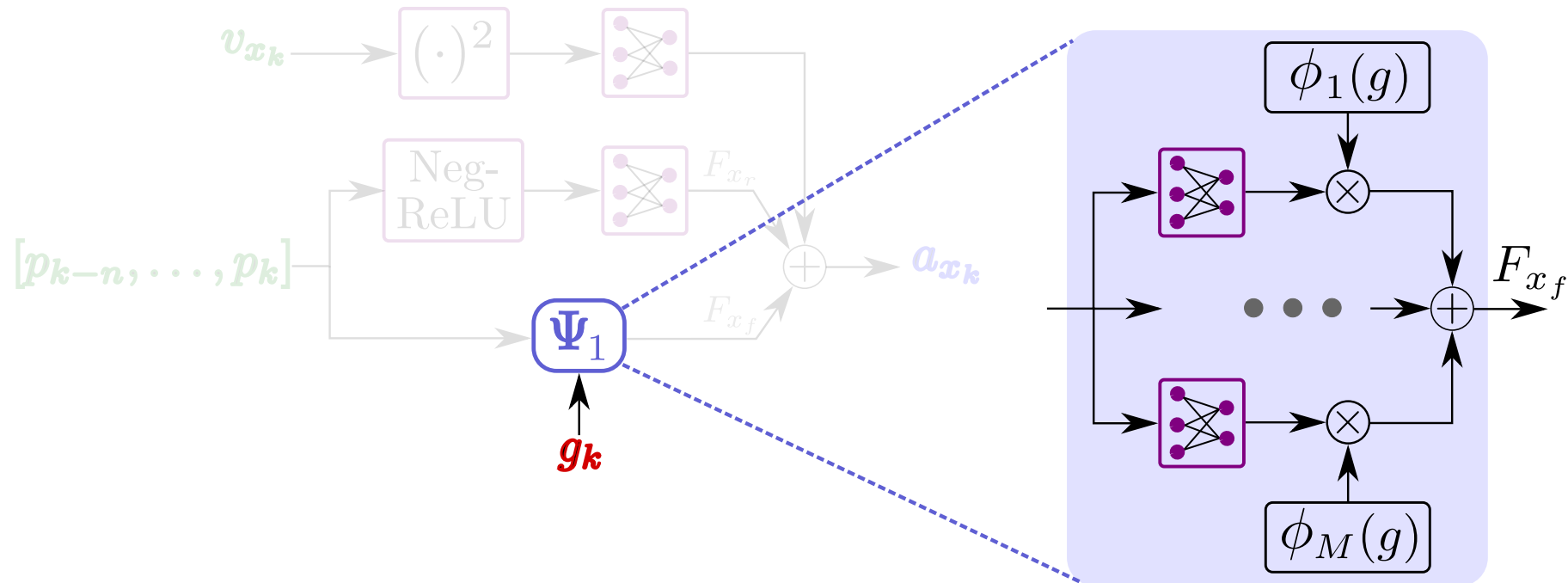
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p, g) + F_{x_r}(p) - k_d v_x^2 \right]$$



Dynamics Learning – Longitudinal Dynamics

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

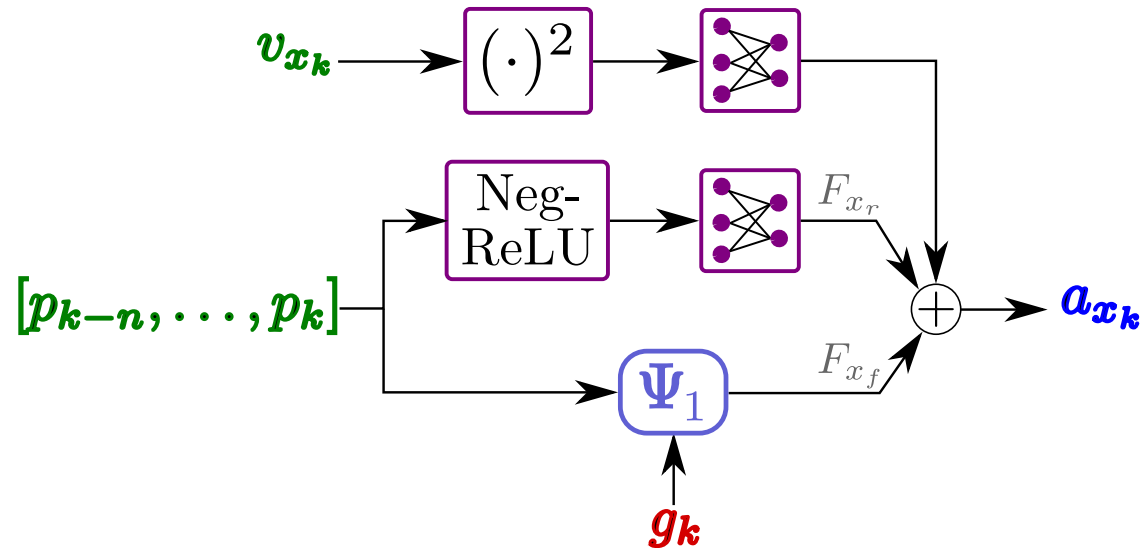
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p, g) + F_{x_r}(p) - k_d v_x^2 \right]$$



Dynamics Learning – Longitudinal Dynamics

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

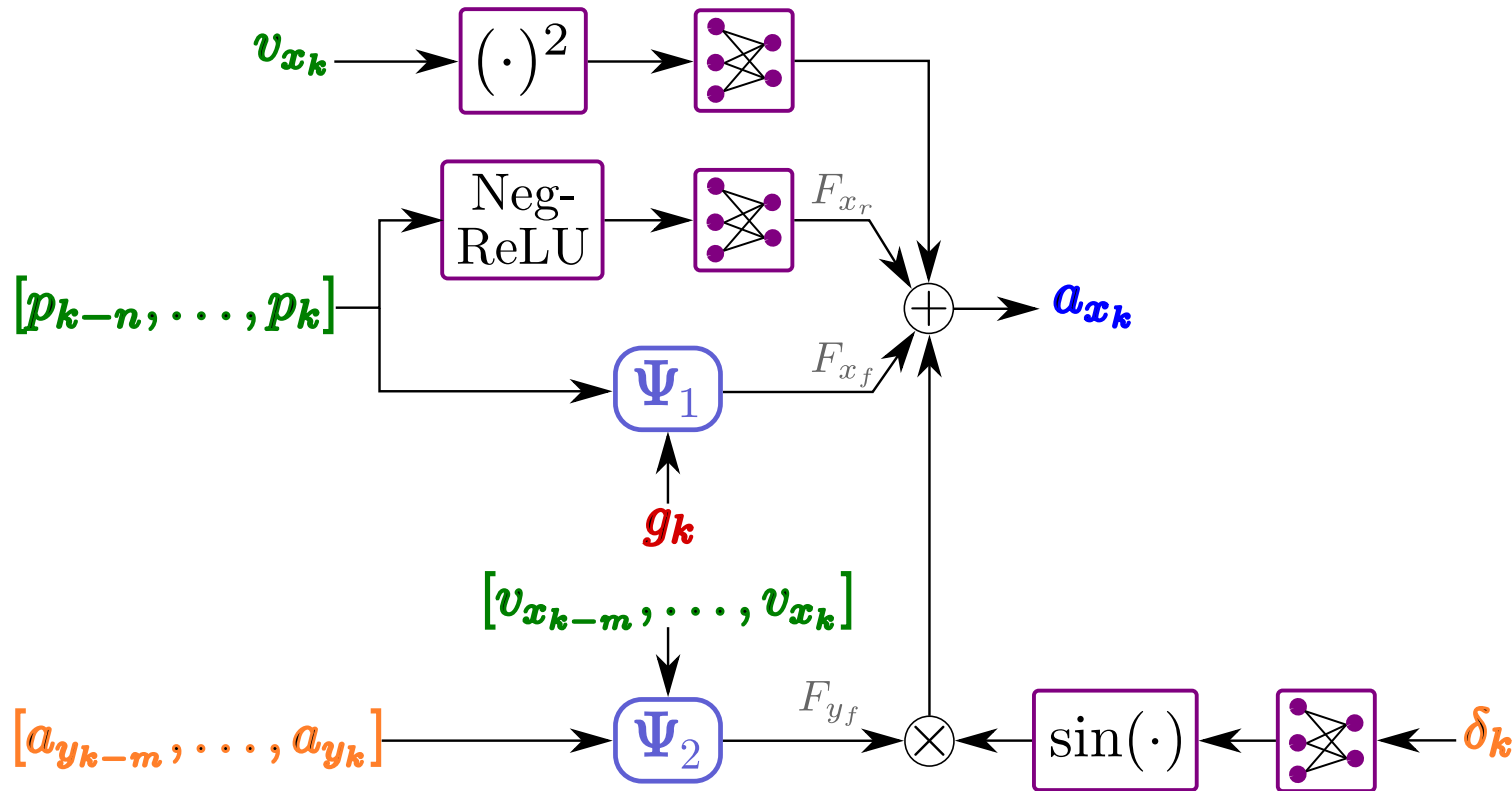
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p, g) + F_{x_r}(p) - k_d v_x^2 \right]$$



Dynamics Learning – Longitudinal Dynamics

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

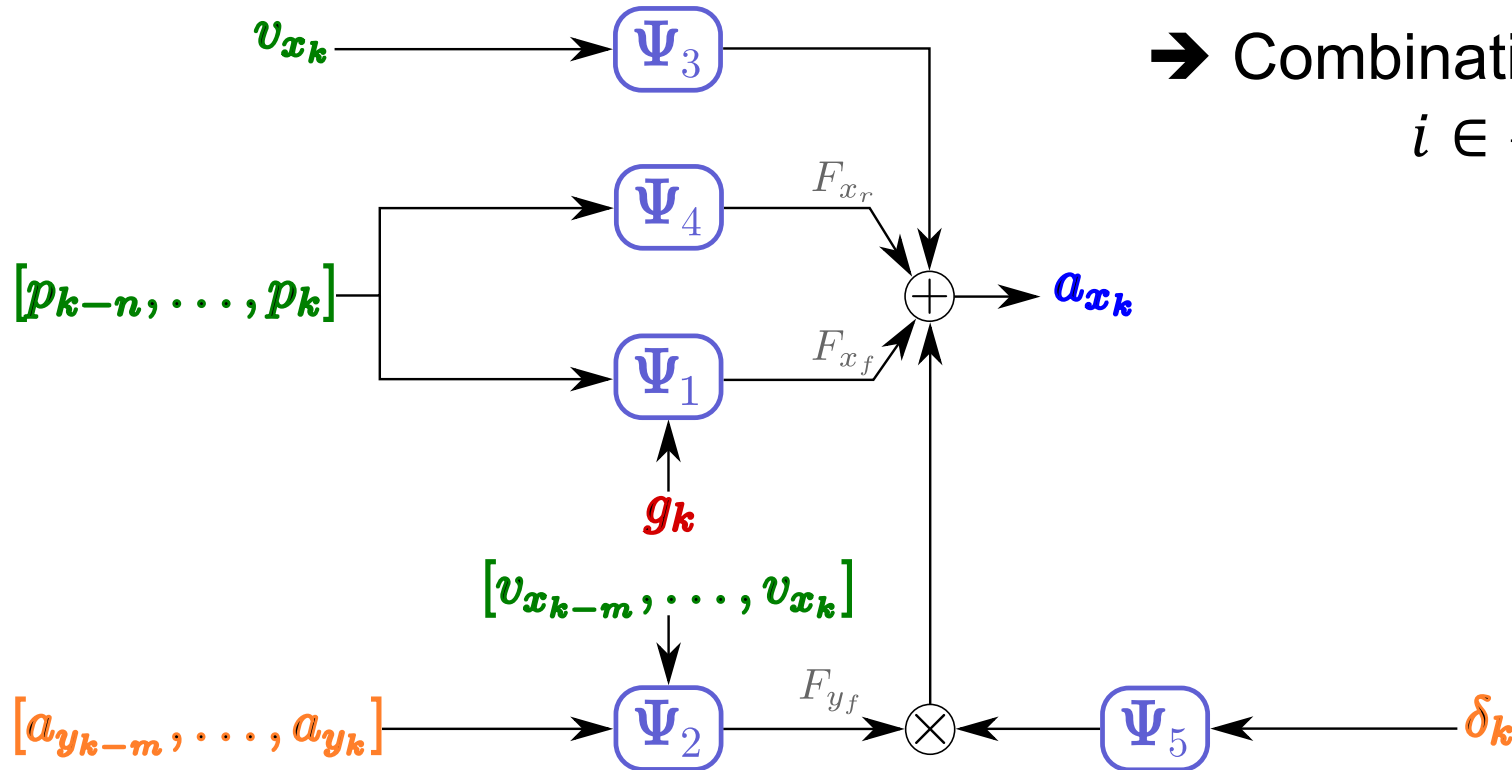
$$a_x = \frac{1}{m} \sum_i F_i = \frac{1}{m} \left[F_{x_f}(p, g) + F_{x_r}(p) - k_d v_x^2 - F_{y_f} \sin(\delta) \right]$$



Dynamics Learning – Longitudinal Dynamics

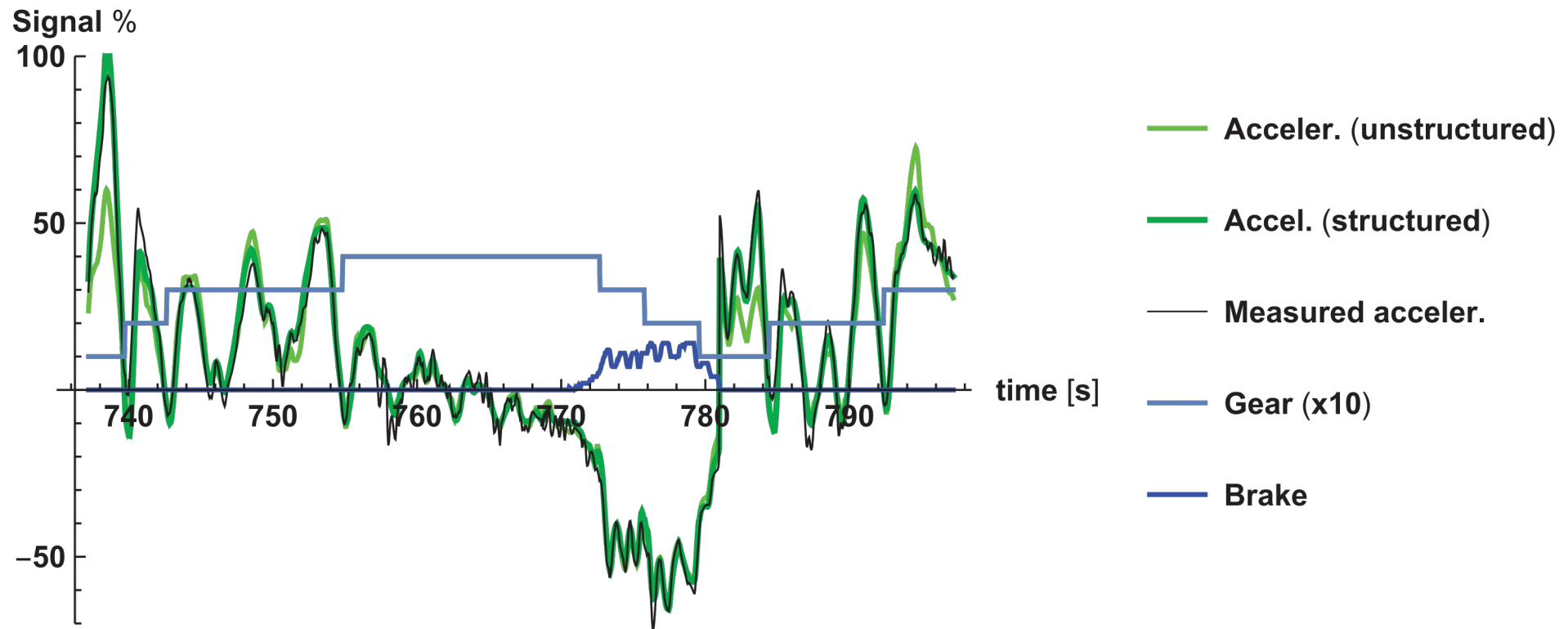
- **Inputs:** pedal $p \in [-1,1]$, speed v_x , gear g , steer δ , lateral accel. a_y
- **Output:** longitudinal accel. a_x

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



→ Combination of **NCMs** Ψ_i ,
 $i \in \{1, \dots, 5\}$

Dynamics Learning – Longitudinal Dynamics

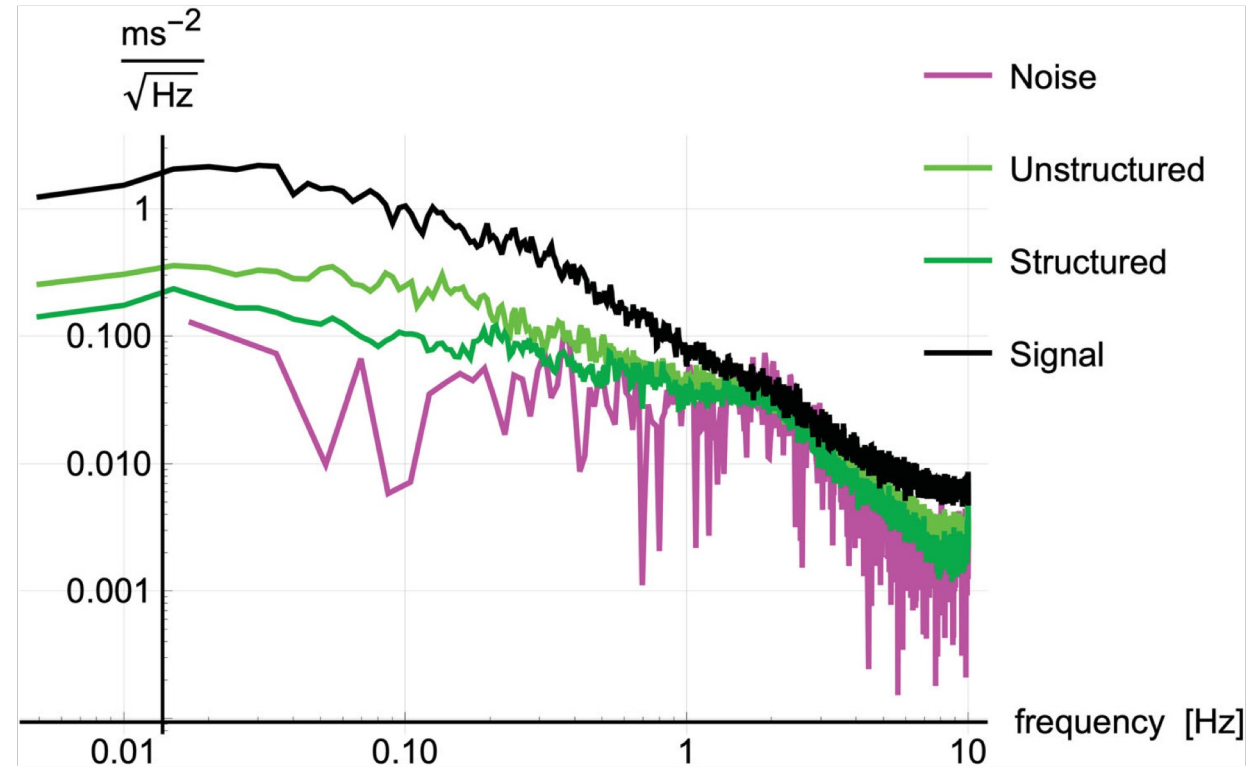


MS-NN (“structured”) versus General-Purpose NN (“unstructured”):

MS-NN is **more accurate on unseen data**

Dynamics Learning – Longitudinal Dynamics

Power Spectral Densities

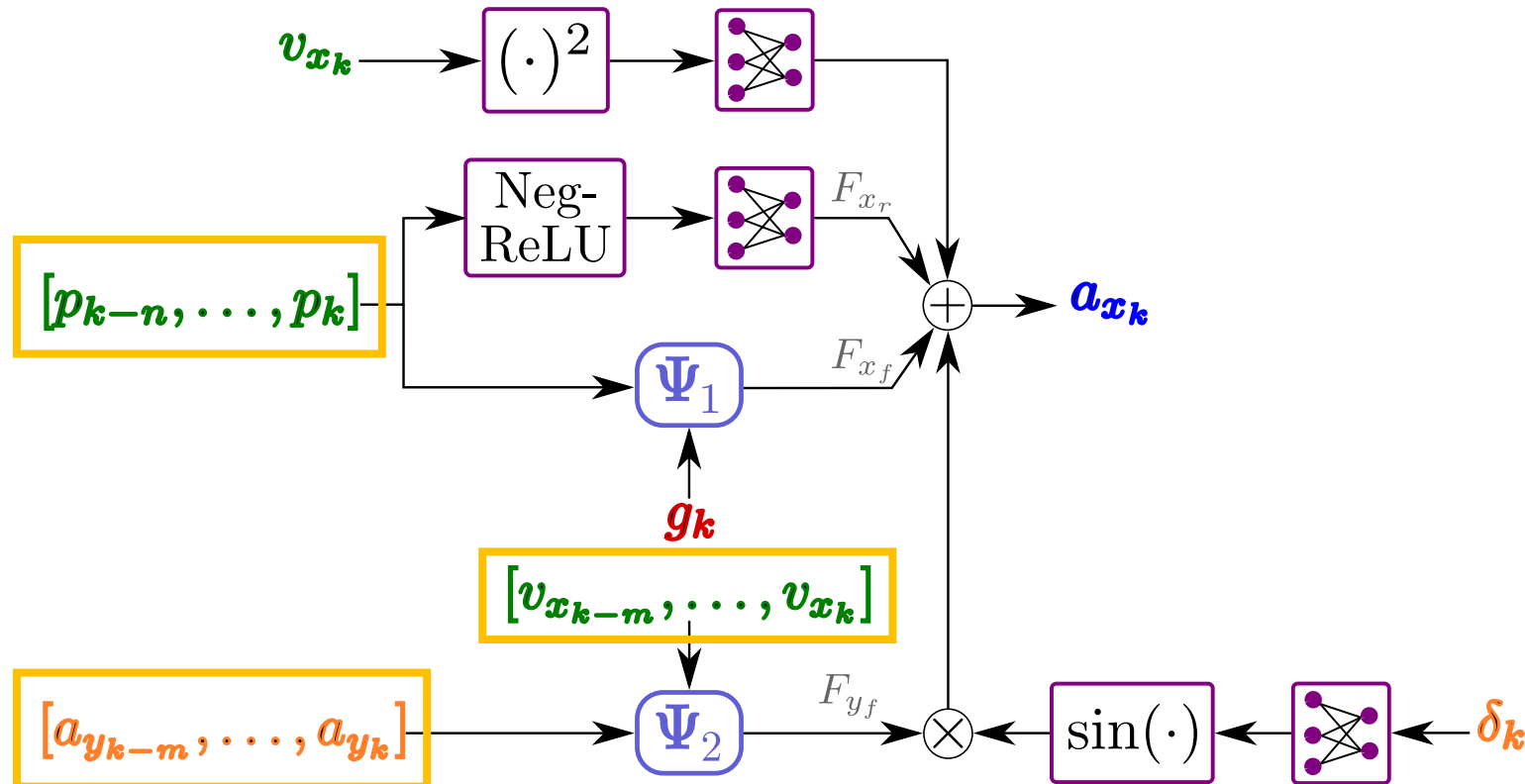


- MS-NN's residuals have lower power → better dynamics learning
- Signal-to-noise drops after 2-3 Hz → Only noise / vibrations beyond 3 Hz

Can we exploit this in the **MS-NN design**?

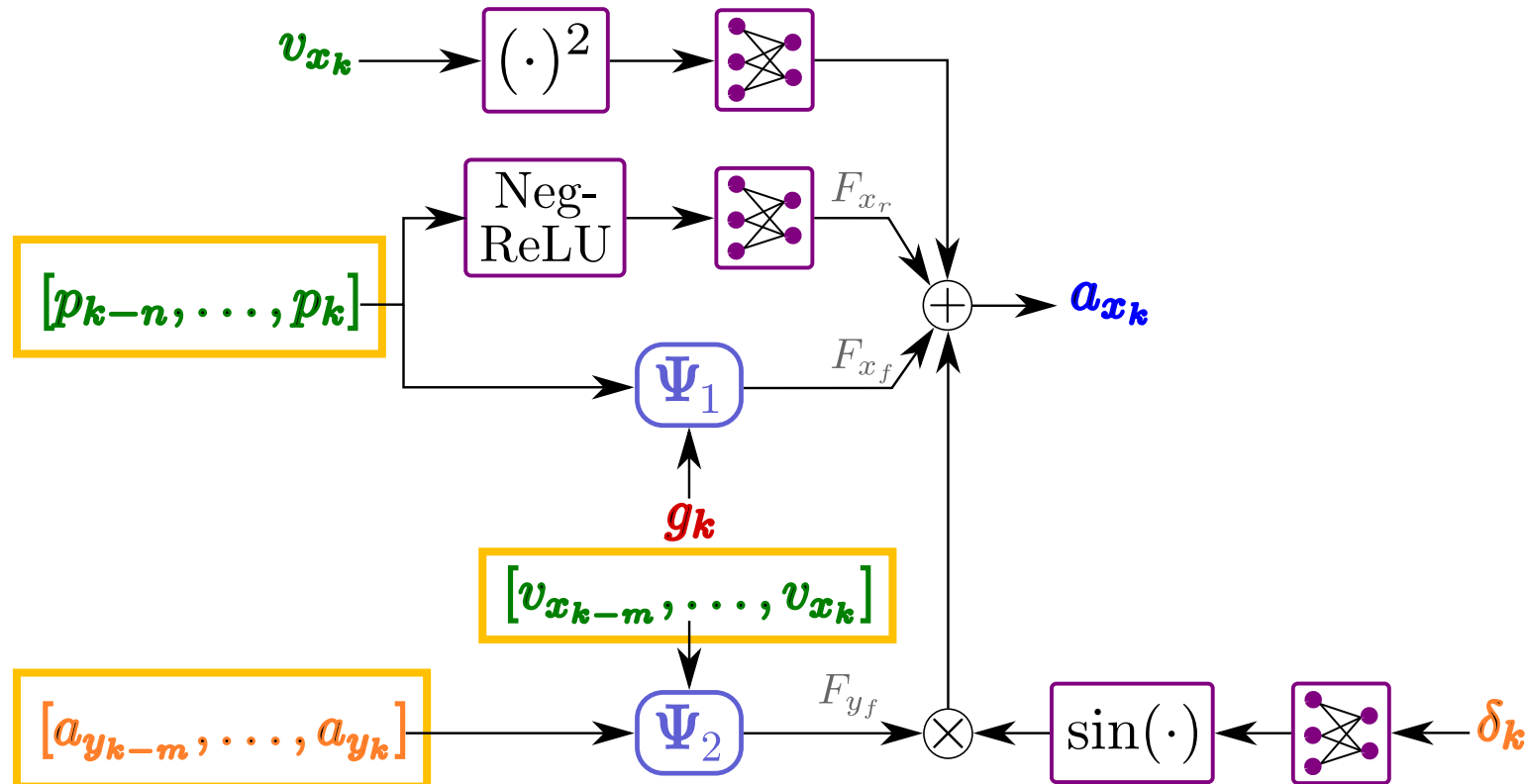
Dynamics Learning – Longitudinal Dynamics

- **Input windows length:** If our frequency content is concentrated in 1–3 Hz (0.33 s), then a past history of $3 - 5 \cdot 0.33 \approx 1 - 1.5$ s is enough for dynamics learning
- We select $n = m = 25$ and a sampling time of 0.05 s $\Rightarrow n \cdot T_s = 1.25$ s



Dynamics Learning – Longitudinal Dynamics

- In principle, we can design n different from m based on the **time-scales** of the longitudinal and lateral vehicle dynamics
- For higher values of n and m we **risk overfitting** „non-physical“ information!



Dynamics Learning – Longitudinal Dynamics



Implementing this in **NNode** is very simple: you find it [here](#)!

```
#Create neural model inputs
velocity = Input('vel')
brake = Input('brk')
gear = Input('gear')
torque = Input('trq')
altitude = Input('alt', dimensions=na)
acc = Input('acc')

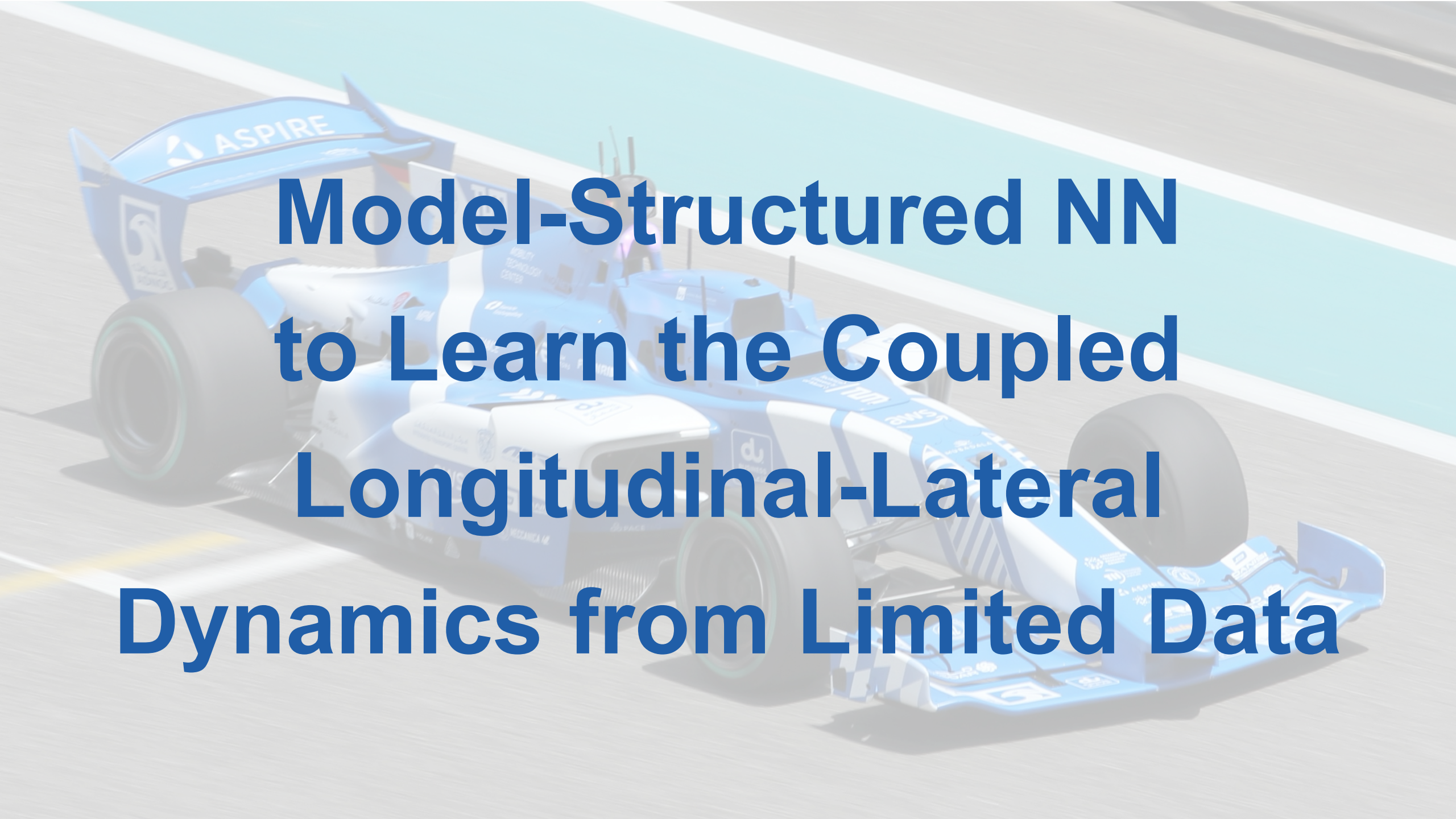
# Create neural network relations
air_drag_force = Linear(b=True)(velocity.last()*2)
breaking_force = -Relu(Fir(W_init = 'init_negexp', W_init_params={'size_index':0, 'first_value':0.002, 'lambda':3})(brake.sw(n)))
gravity_force = Linear(W_init='init_constant', W_init_params={'value':0}, dropout=0.1, W='gravity')(altitude.last())
fuzzi_gear = Fuzzify(6, range=[2,7], functions='Rectangular')(gear.last())
local_model = LocalModel(input_function=lambda: Fir(W_init = 'init_negexp', W_init_params={'size_index':0, 'first_value':0.002, 'lambda':3}))
engine_force = local_model(torque.sw(n), fuzzi_gear)

# Create neural network output
out = Output('accelleration', air_drag_force+breaking_force+gravity_force+engine_force)

# Add the neural model to the nnode structure and neuralization of the model
vehicle.addModel('acc',[out])
vehicle.addMinimize('acc_error', acc.last(), out, loss_function='rmse')
vehicle.neuralizeModel(0.05)
```

Questions & Discussion

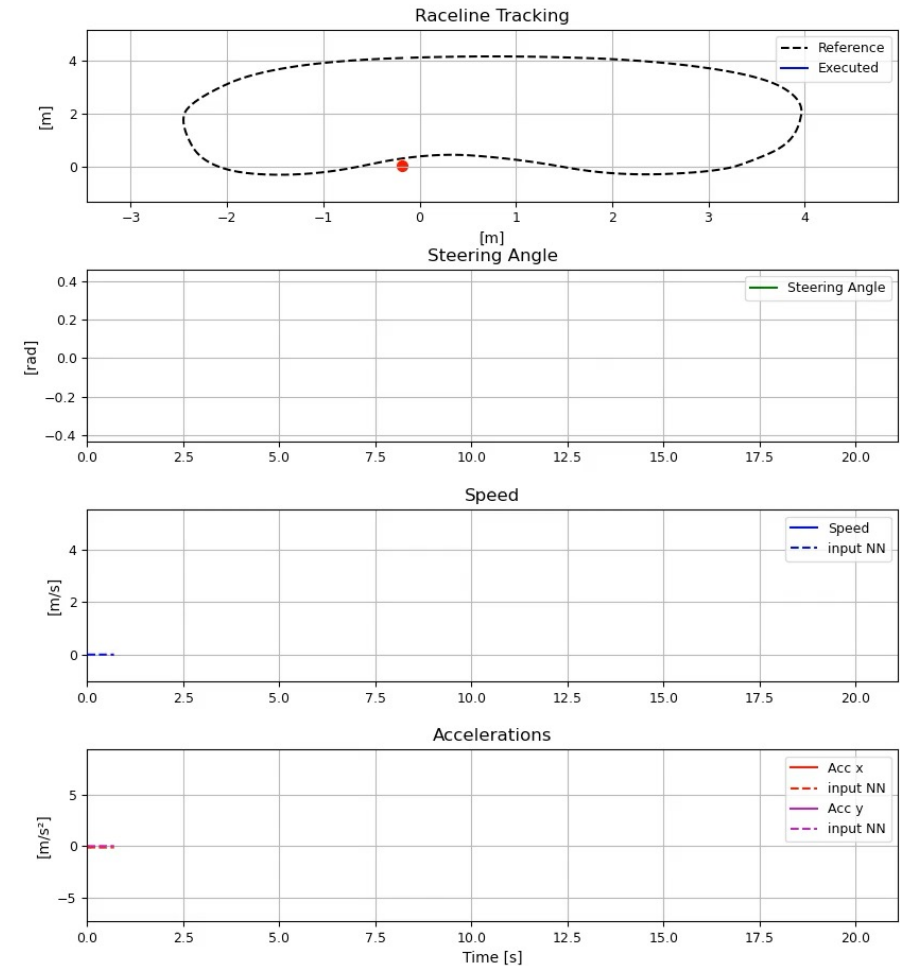
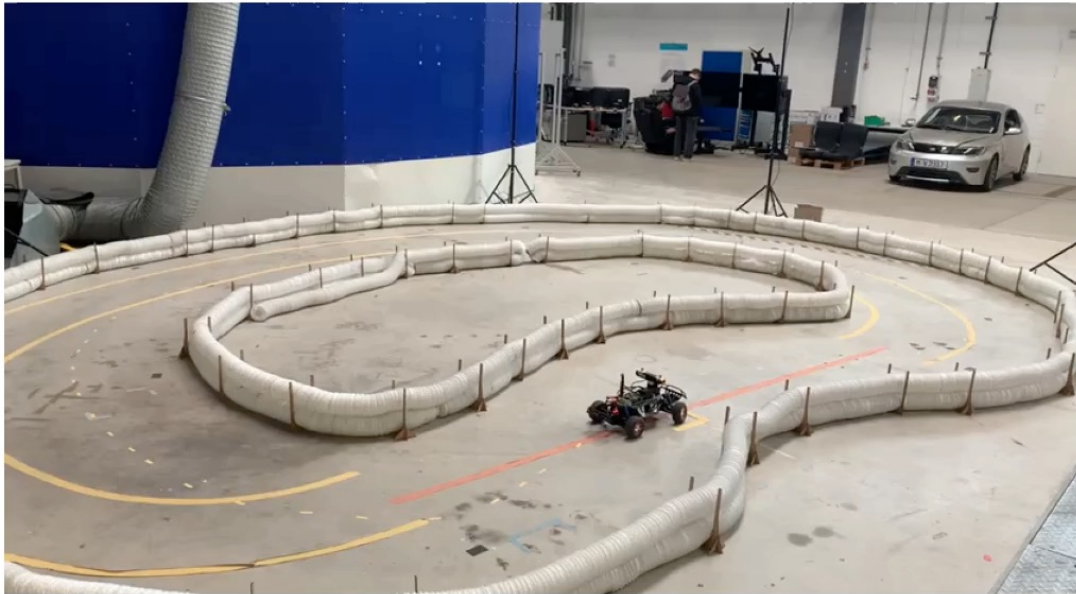


A blue and white Formula 1 car is shown from a side-rear perspective, driving on a grey racetrack with white and yellow markings. The car features various sponsor logos, including 'ASPIRE' on the rear wing, 'AWS' on the sidepods, and 'du' on the front wing. The text 'Model-Structured NN to Learn the Coupled Longitudinal-Lateral Dynamics from Limited Data' is overlaid in a large, bold, blue font across the center of the image.

Model-Structured NN to Learn the Coupled Longitudinal-Lateral Dynamics from Limited Data

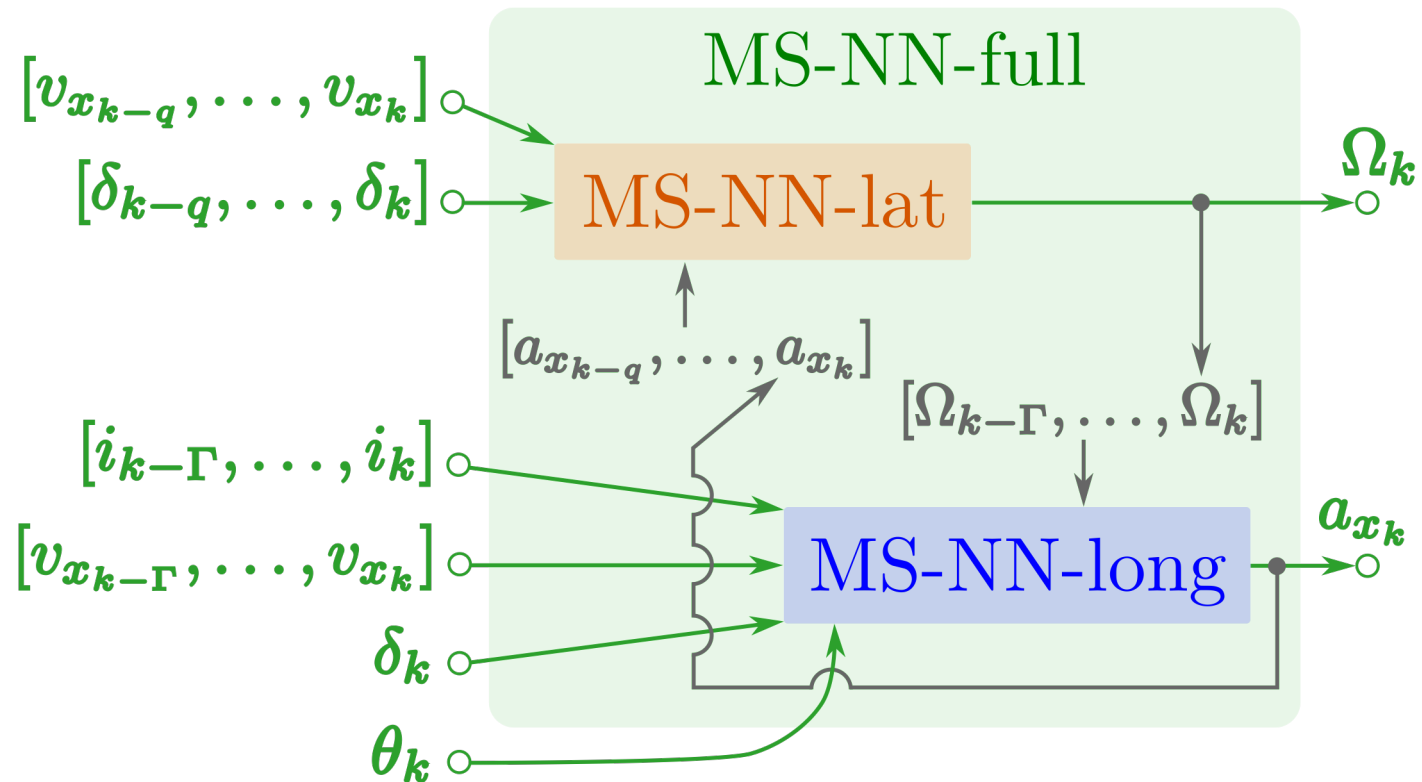
Dynamics Learning – Combined Longit. - Lateral Dynamics

- Combined nonlinear longit.-lateral dynamics learning near the vehicle limits



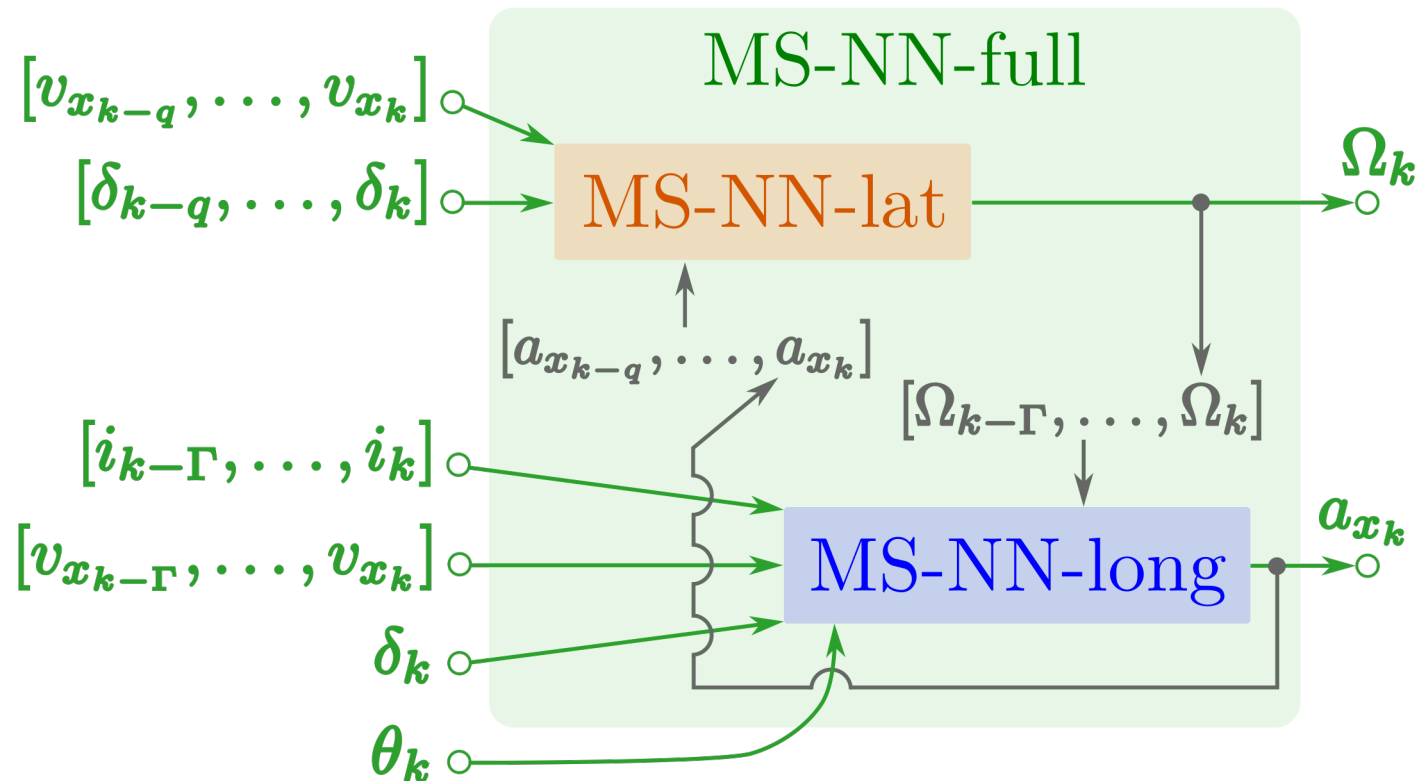
Dynamics Learning – Combined Longit. - Lateral Dynamics

- **MS-NN-long** to learn the combined longitudinal dynamics
- **MS-NN-lat** to learn the combined lateral dynamics



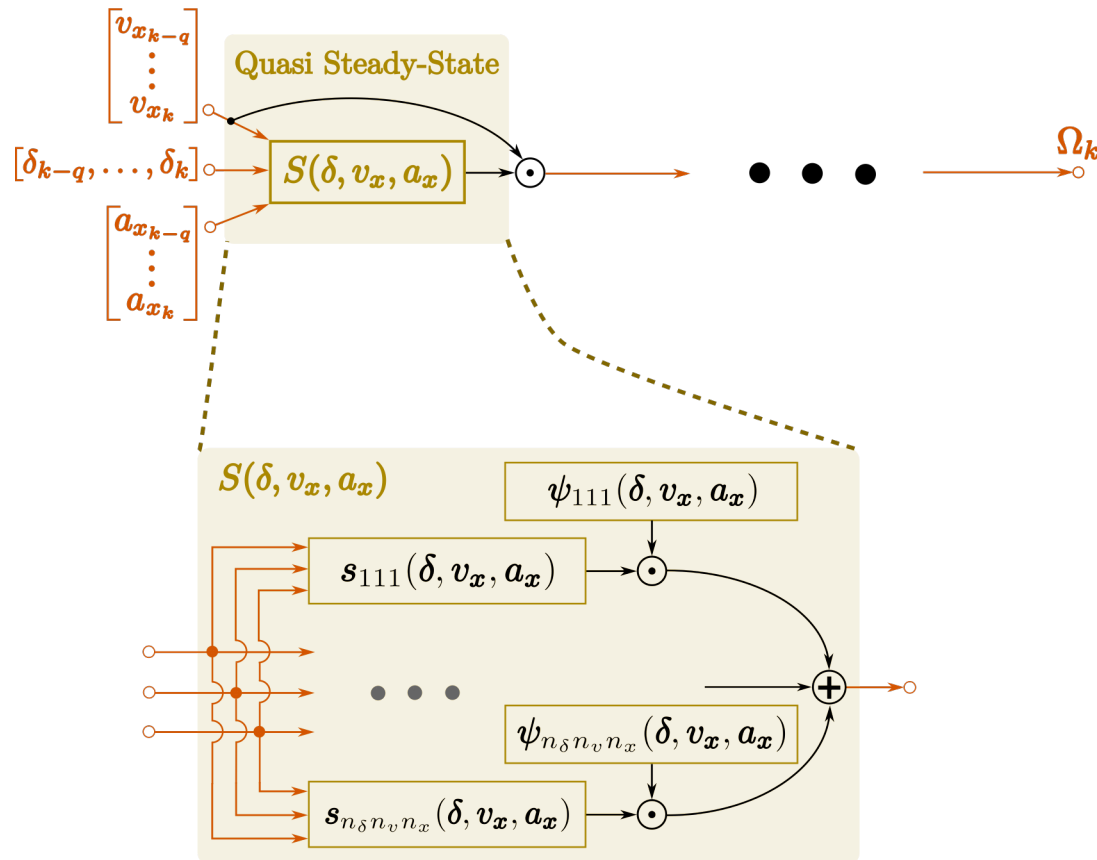
Dynamics Learning – Combined Longit. - Lateral Dynamics

- **Inputs:** speed (v_x), steer angle (δ), motor current (i), road slope (θ)
- **Outputs:** yaw rate (Ω), longitudinal acceleration (a_x)

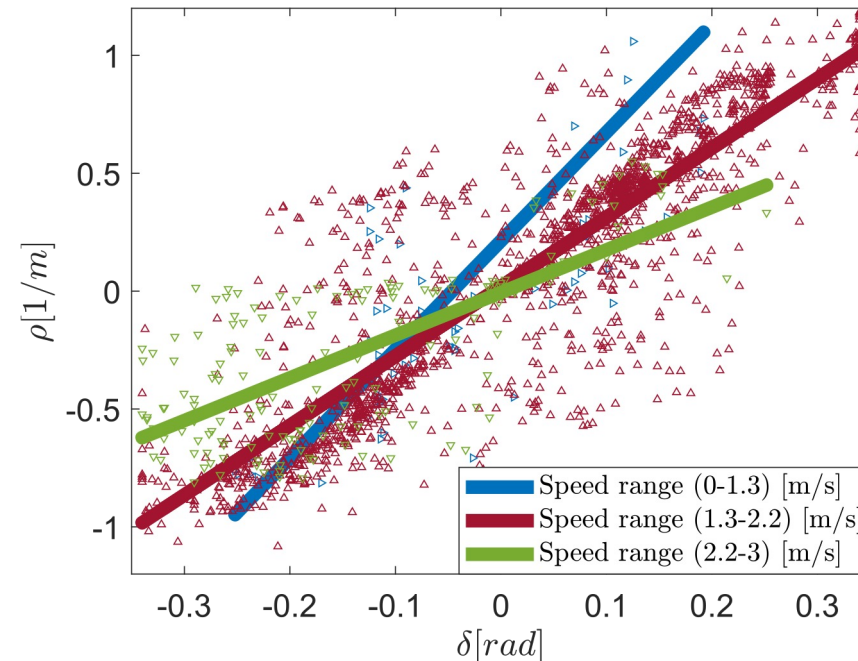


Dynamics Learning – Combined Longit. - Lateral Dynamics

MS-NN-lat to learn the combined lateral dynamics

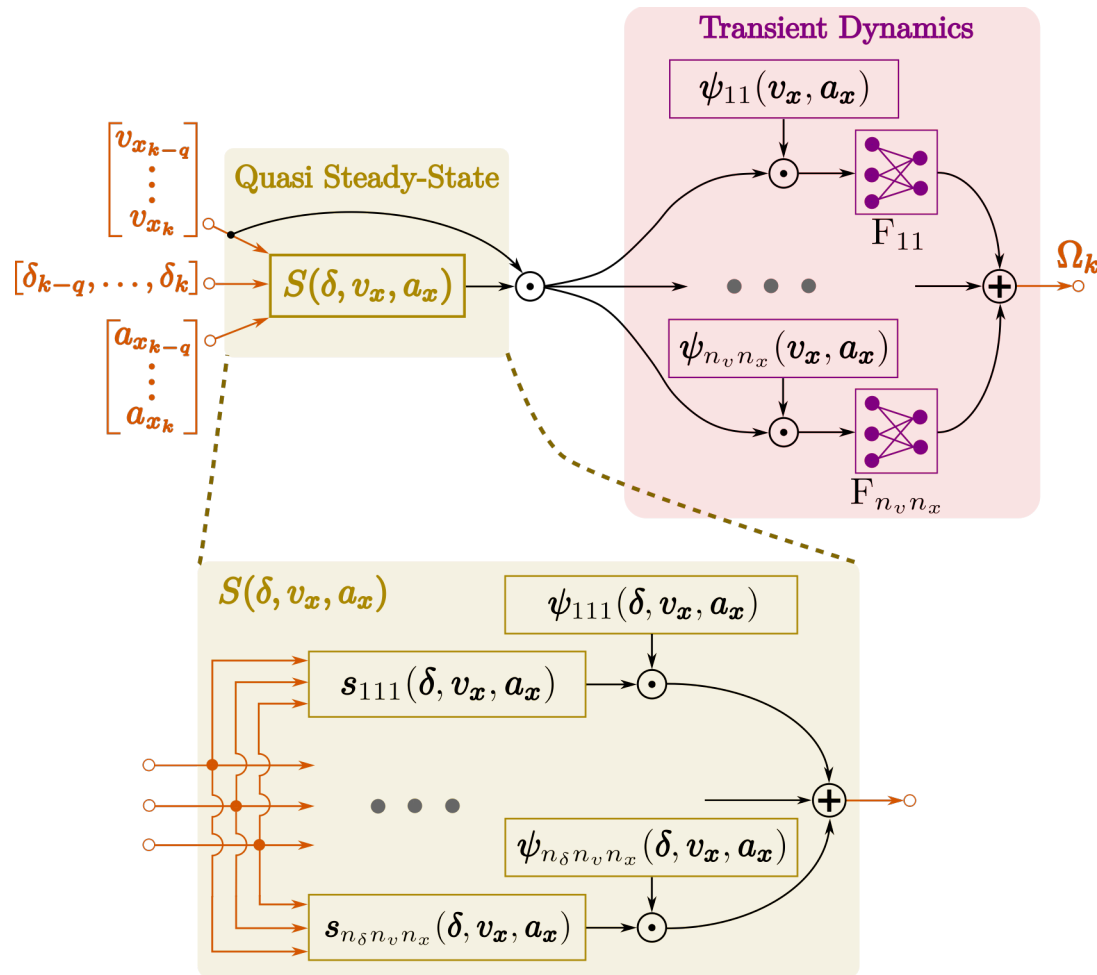


- Neural compositional modules to learn the **quasi steady-state** behavior

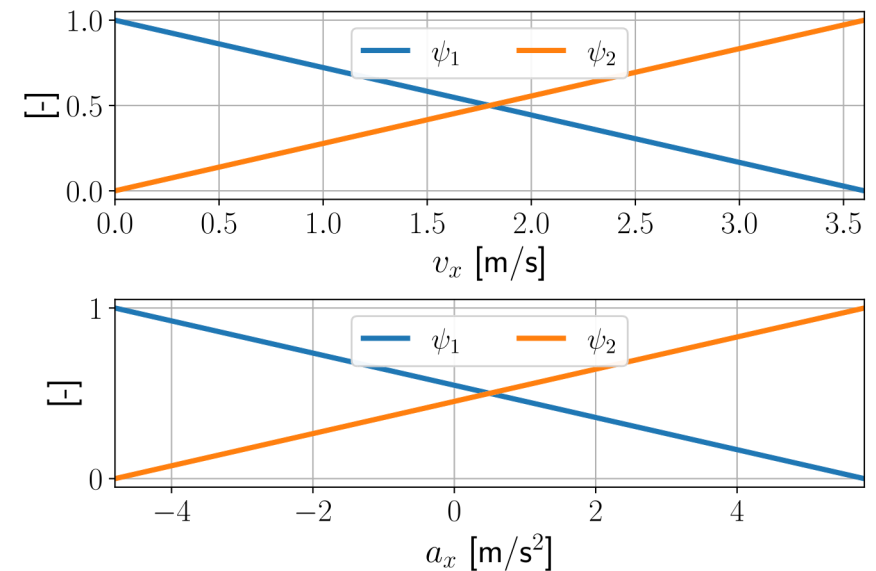


Dynamics Learning – Combined Longit. - Lateral Dynamics

MS-NN-lat to learn the combined lateral dynamics

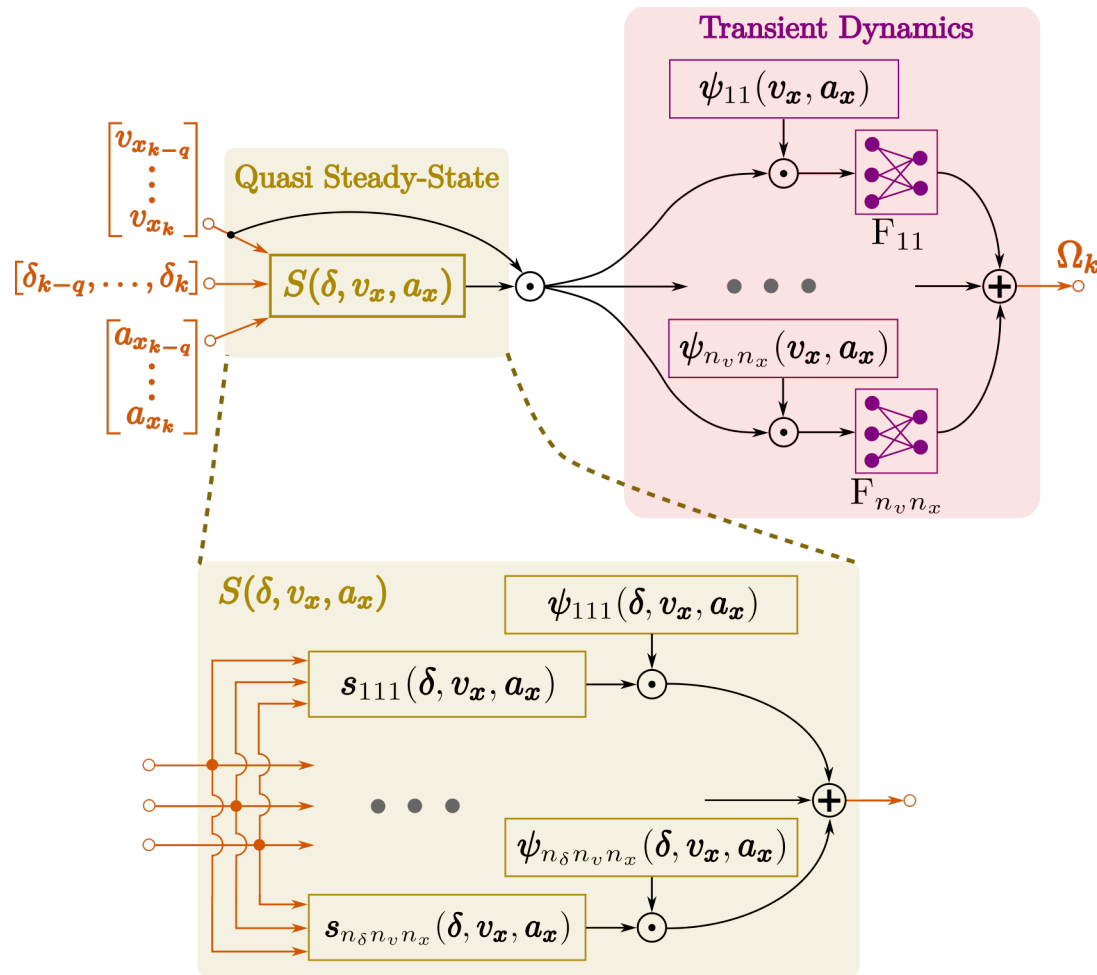


- NCMs to learn the **transient dynamics** → Newton's laws say they depend on v_x , a_x , so we can leverage this information

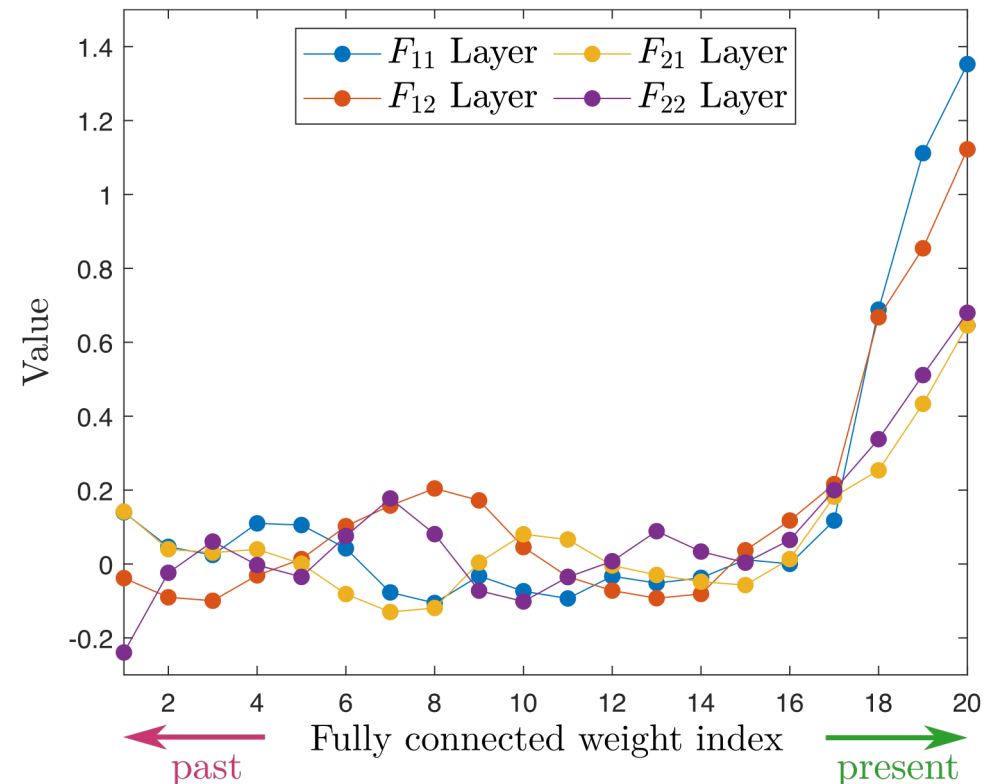


Dynamics Learning – Combined Longit. - Lateral Dynamics

MS-NN-lat to learn the combined lateral dynamics



- Can we interpret the **FIR layers**? YES!



Dynamics Learning – Combined Longit. - Lateral Dynamics

Better generalization with **small training sets** than **general-purpose NN (G-NN)**

Models	Metrics	Large Train. Set		Medium Train. Set		Small Train. Set	
		Train.	Valid.	Train.	Valid.	Train.	Valid.
G-NN	RMSE ↓ [rad/s]	0.102	0.142	0.135	0.172	0.134	0.224
	FVU ↓ [-]	0.022	0.053	0.030	0.072	0.037	0.155
MS-NN-bench [15]	RMSE ↓ [rad/s]	0.147	0.154	0.159	0.158	0.181	0.181
	FVU ↓ [-]	0.056	0.060	0.059	0.063	0.088	0.066
MS-NN-Ablation 1	RMSE ↓ [rad/s]	<u>0.081</u>	0.122	<u>0.079</u>	<u>0.110</u>	<u>0.088</u>	<u>0.134</u>
	FVU ↓ [-]	<u>0.014</u>	0.033	<u>0.012</u>	<u>0.032</u>	<u>0.019</u>	<u>0.053</u>
MS-NN-Ablation 2	RMSE ↓ [rad/s]	0.085	<u>0.116</u>	0.084	0.116	0.094	0.124
	FVU ↓ [-]	0.016	<u>0.036</u>	0.014	0.035	0.022	0.041
MS-NN-full (ours)	RMSE ↓ [rad/s]	0.076	0.109	0.076	0.106	0.085	0.137
	FVU ↓ [-]	0.012	0.031	0.012	0.030	0.019	0.061

Dynamics Learning – Combined Longit. - Lateral Dynamics

Quick adaptation to new conditions by **fine-tuning specific sub-modules**

- **20 sec** of new data: different tires / changed mass
- Selective fine-tuning of lateral and longitudinal dynamics parameters

Test Case	Models	Yaw Rate Ω : RMSE (FVU)		Long. Accelerat. a_x : RMSE (FVU)	
		Zero-shot	Fine-tuning	Zero-shot	Fine-tuning
Tire set 1	G-NN	0.147 (0.050)	→ 0.107 (0.027)	0.614 (0.485)	→ 0.380 (0.185)
	MS-NN-bench	0.102 (0.024)	→ <u>0.096 (0.022)</u>	0.462 (0.275)	→ 0.425 (0.233)
	MS-NN-full (ours)	0.172 (0.068)	→ 0.085 (0.017)	0.531 (0.363)	→ <u>0.395 (0.201)</u>
Tire set 2	G-NN	0.110 (0.067)	→ 0.070 (0.027)	0.624 (0.537)	→ 0.368 (0.187)
	MS-NN-bench	0.079 (0.035)	→ <u>0.059 (0.019)</u>	0.373 (0.191)	→ <u>0.347 (0.166)</u>
	MS-NN-full (ours)	0.129 (0.093)	→ 0.053 (0.015)	0.468 (0.300)	→ 0.334 (0.153)
+10% Mass	G-NN	0.088 (0.061)	→ <u>0.046 (0.017)</u>	0.489 (0.974)	→ <u>0.329 (0.441)</u>
	MS-NN-bench	0.072 (0.042)	→ 0.053 (0.023)	0.396 (0.639)	→ 0.375 (0.572)
	MS-NN-full (ours)	0.075 (0.046)	→ 0.045 (0.016)	0.398 (0.648)	→ 0.290 (0.344)

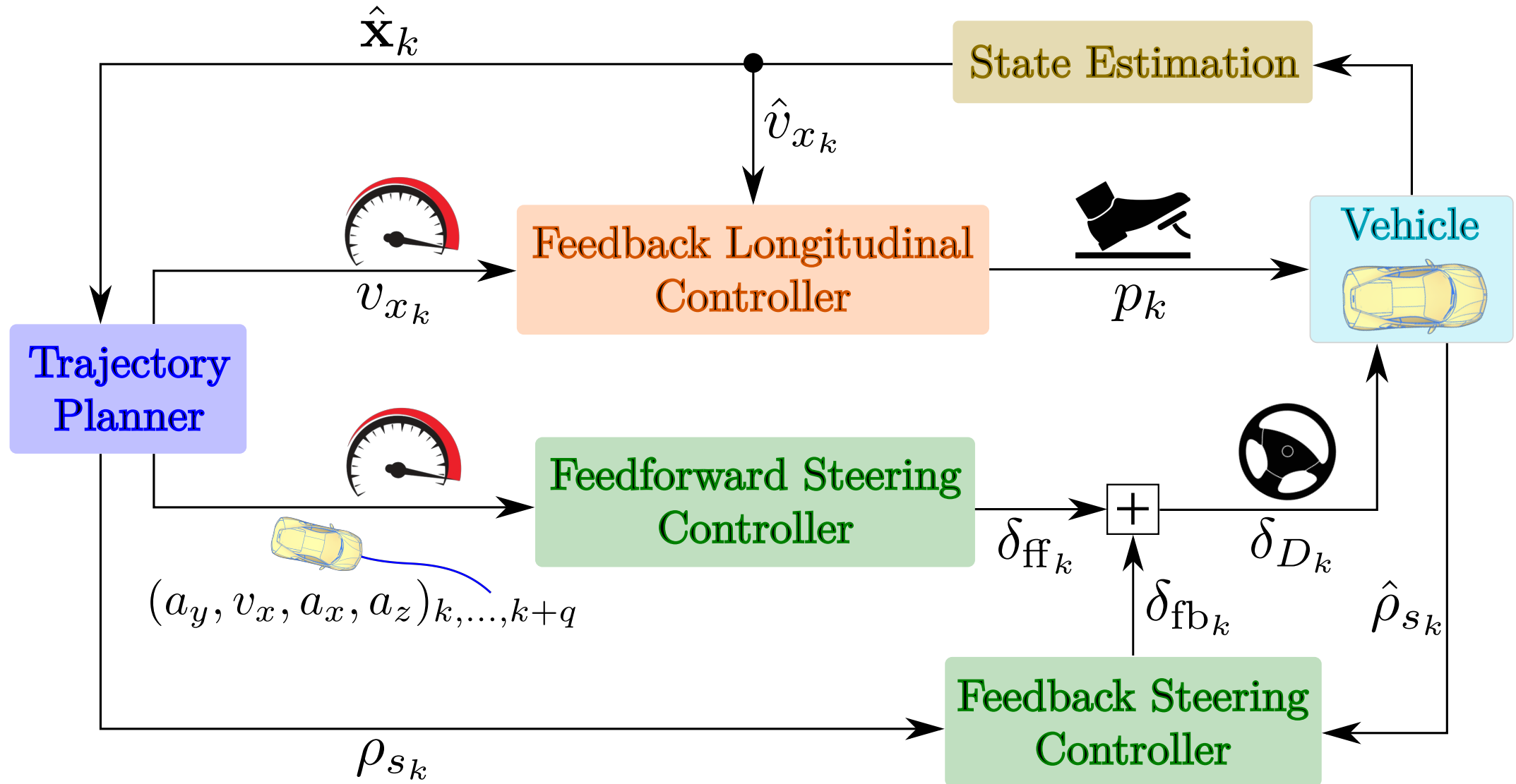


Dynamics Learning – Combined Longit. - Lateral Dynamics

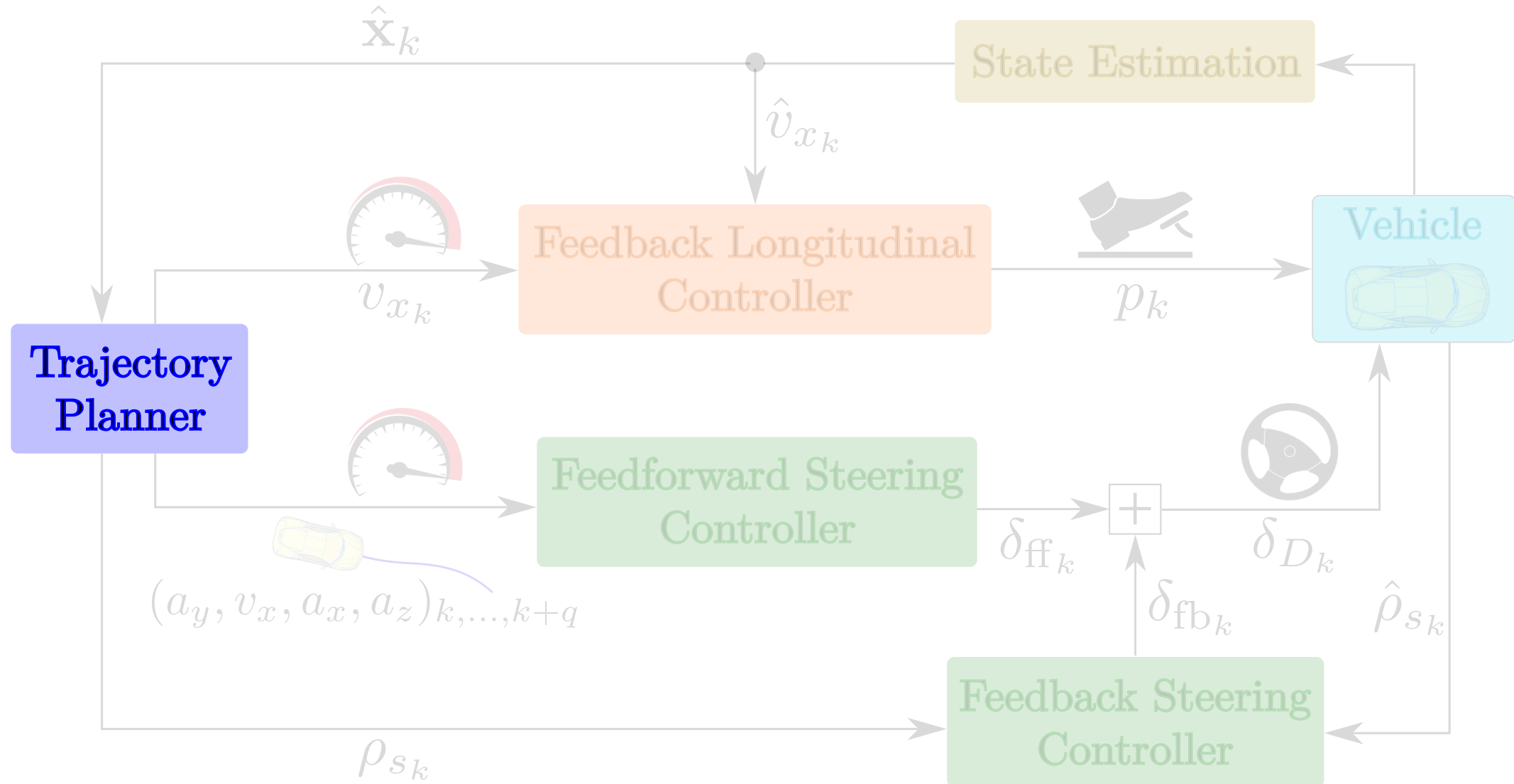
Implementing this in **NModely** is very simple: you find it [here](#)!

The logo for NModely, featuring a stylized lowercase 'm' in a blue-to-black gradient, followed by the word 'odely' in a black sans-serif font.

Motion Generation, Control & Dynamics Learning

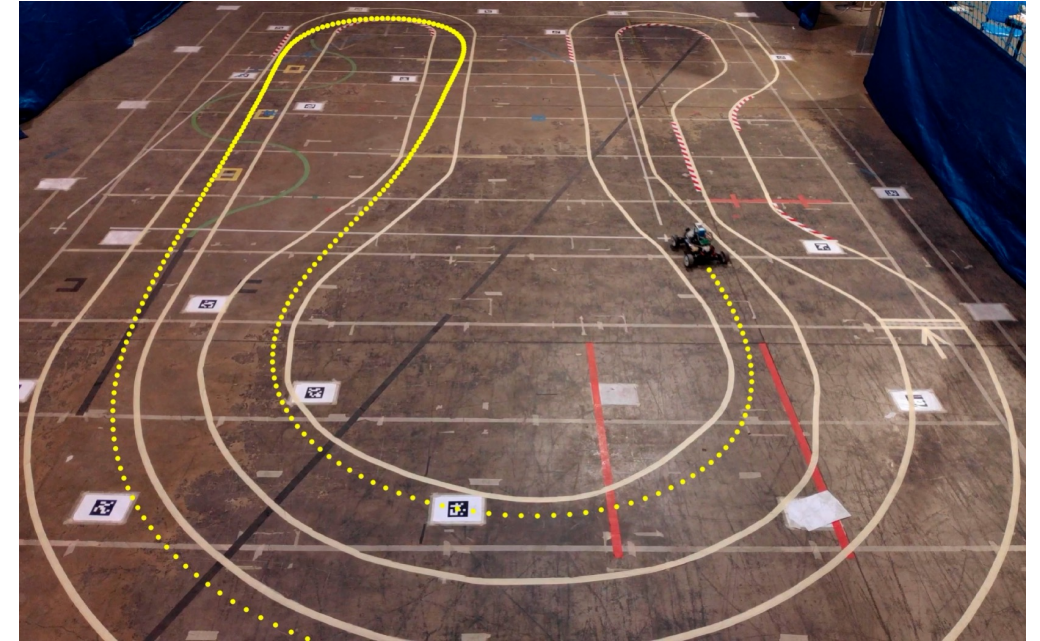
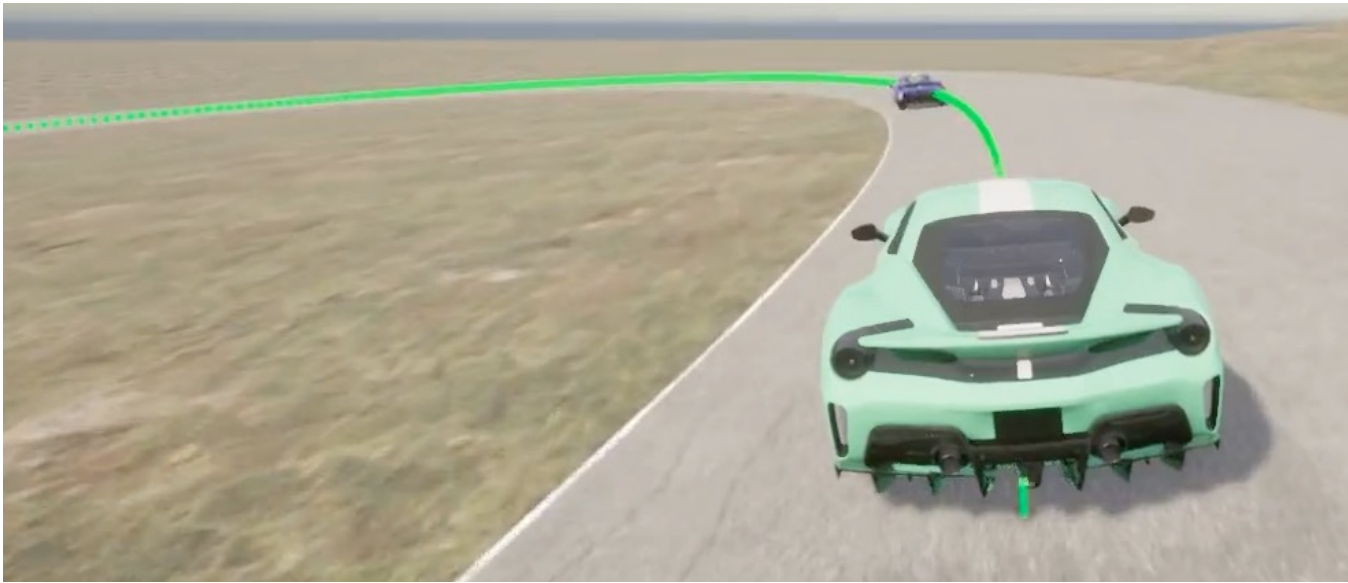


Motion Generation



Motion Generation

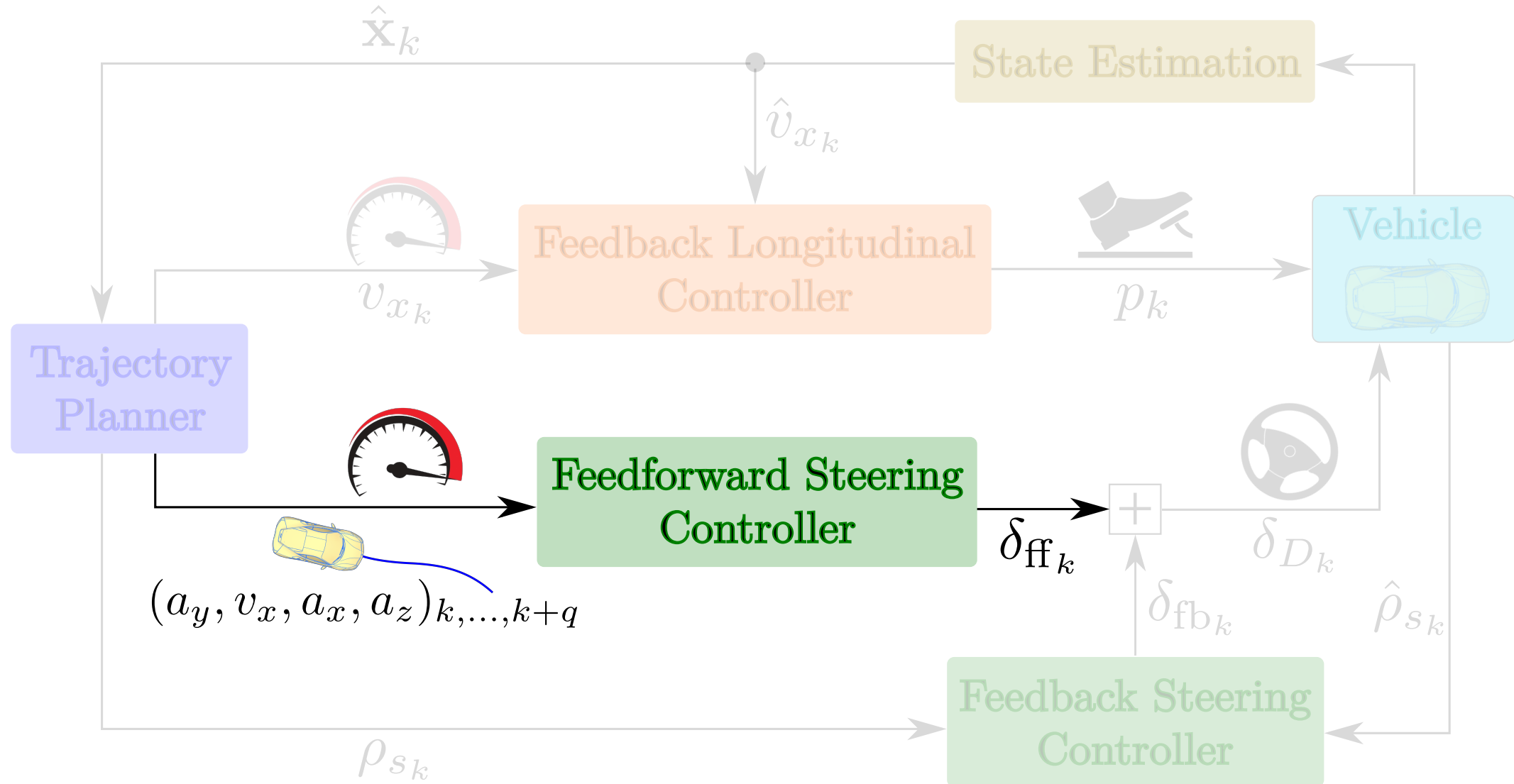
- **Minimum-time** trajectory planning on a long receding horizon
- Capture the **nonlinear** vehicle dynamics and **performance limits**



A blue and white autonomous racing car, branded with 'ASPIRE' and 'TUM', is shown on a racetrack. The car features various sponsor logos including 'AWS', 'd', and 'ASPIRE'. It has a camera mounted on top and is positioned on a track with a blue and white striped curb. The background is a blurred racetrack surface.

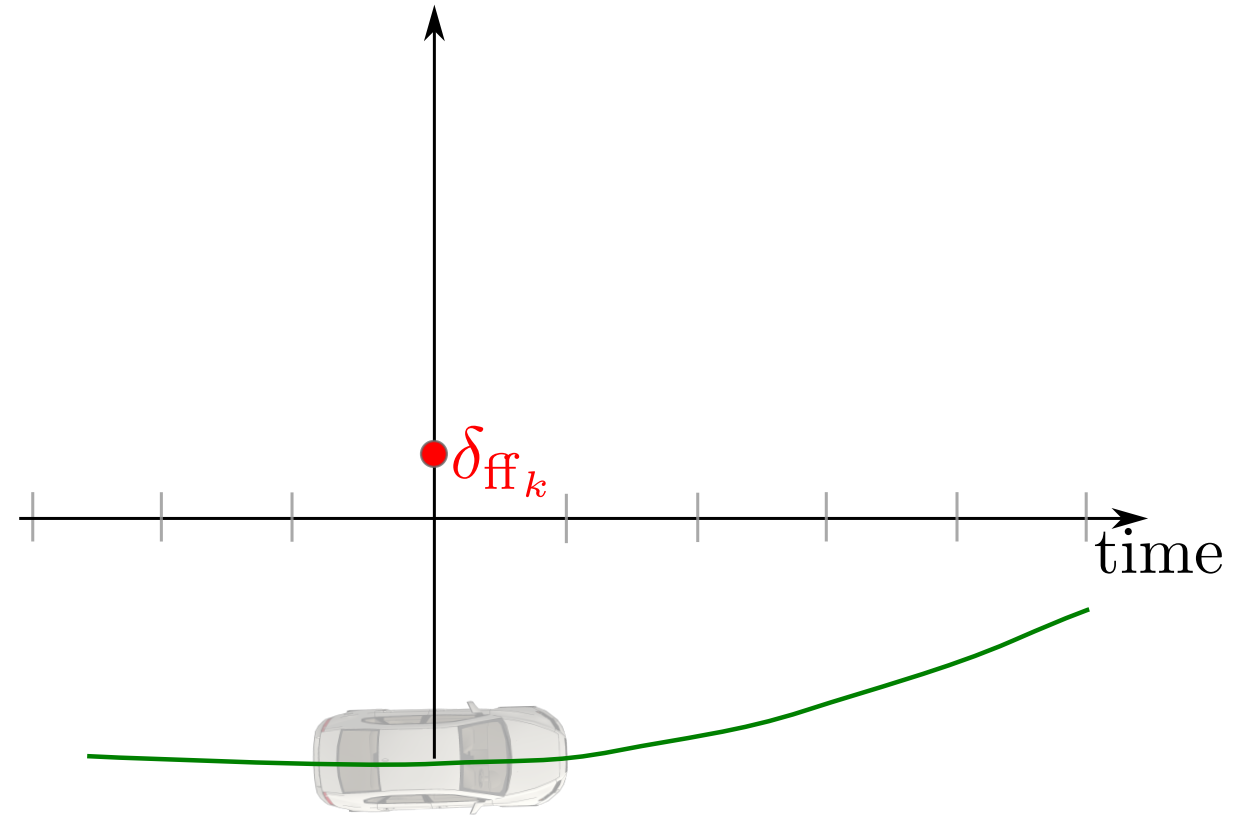
How Do We Execute the Planned Trajectories?

Steering Control



Steering Control

- **Output:** steering wheel angle δ_{ff_k}



Steering Control

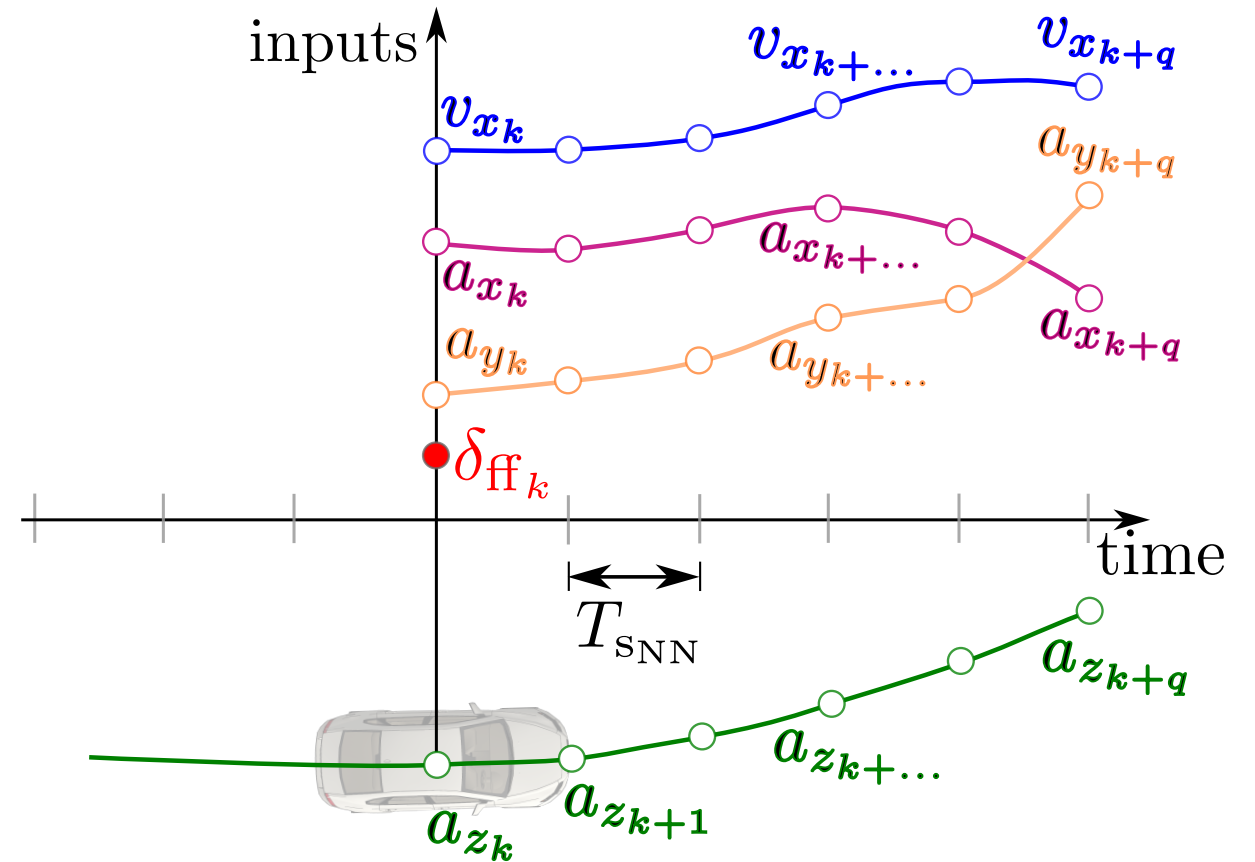
- **Output:** steering wheel angle δ_{ff_k}
- **Inputs:** future reference trajectories

○ $\mathbf{a}_{y_k} = [a_{y_k}, \dots, a_{y_{k+q}}]$

○ $\mathbf{a}_{x_k} = [a_{x_k}, \dots, a_{x_{k+q}}]$

○ $\mathbf{a}_{z_k} = [a_{z_k}, \dots, a_{z_{k+q}}]$

○ $\mathbf{v}_{x_k} = [v_{x_k}, \dots, v_{x_{k+q}}]$



Steering Control

- Baseline controller: **Model-based** design

$$\delta_k = \delta_{\text{kin}}(a_{y_k}, v_{x_k}) + \delta_{\text{us}}(a_{y_k}) + \delta_{\text{ax}}(a_{y_k}, a_{x_k}) + \delta_{\text{off}}$$

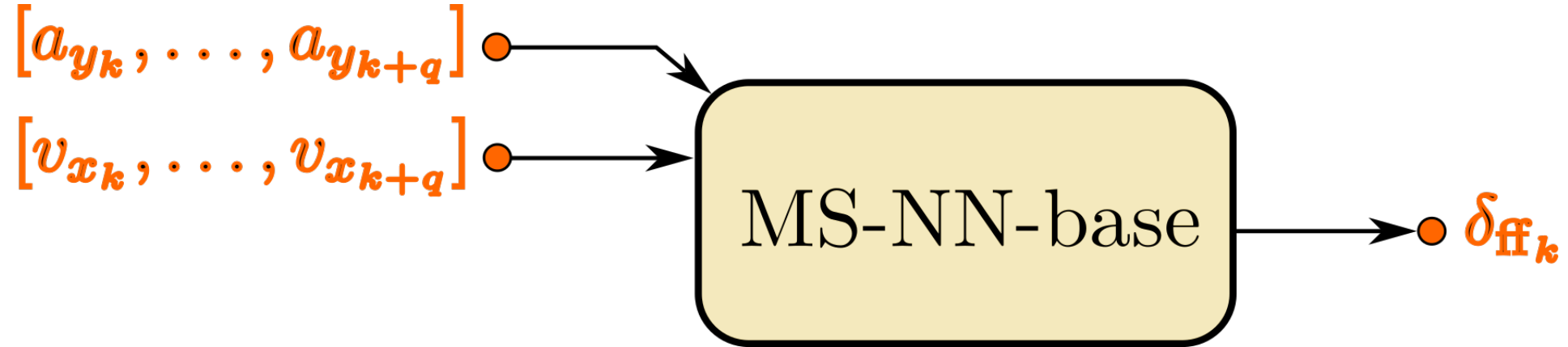
- Simple structure but **limited performance**
- **Idea** to improve: **Model-Structured Neural Network**



Steering Control

Two incremental **model-structured NNs**

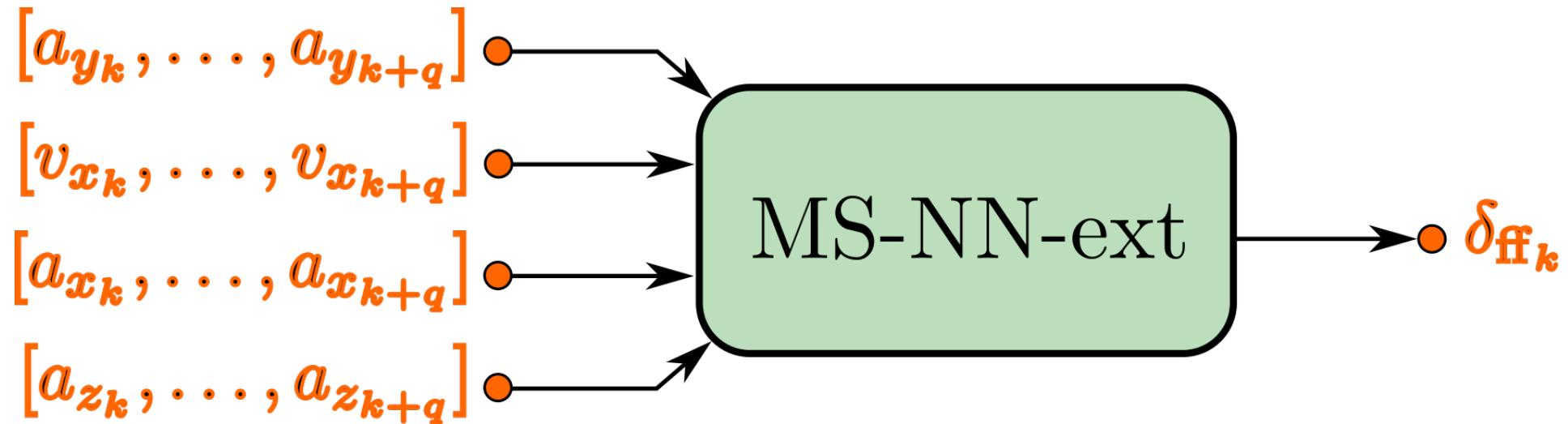
- **MS-NN-base**: pure lateral nonlinear dynamics (2D road)



Steering Control

Two incremental **model-structured NNs**

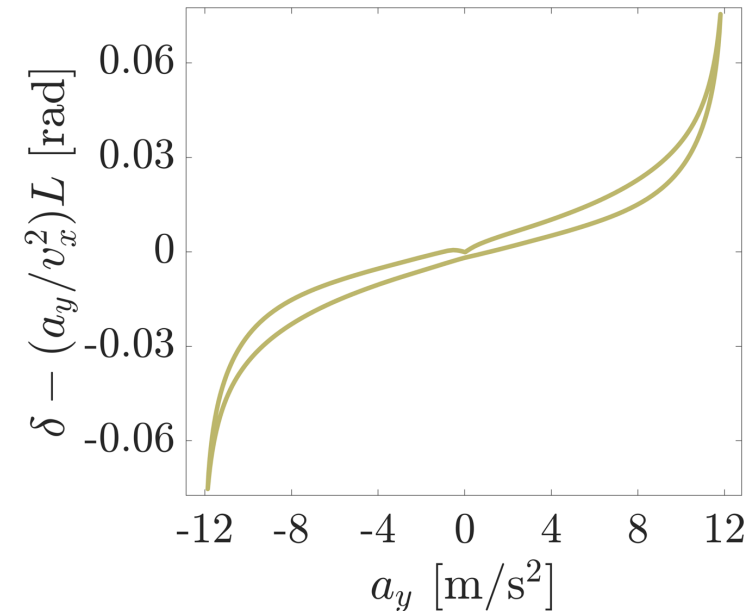
- **MS-NN-base**: pure lateral nonlinear dynamics (2D road)
- **MS-NN-ext**: effects of longitudinal and vertical accelerations (3D road)



Steering Control

MS-NN-base: Steady-state pure lateral dynamics

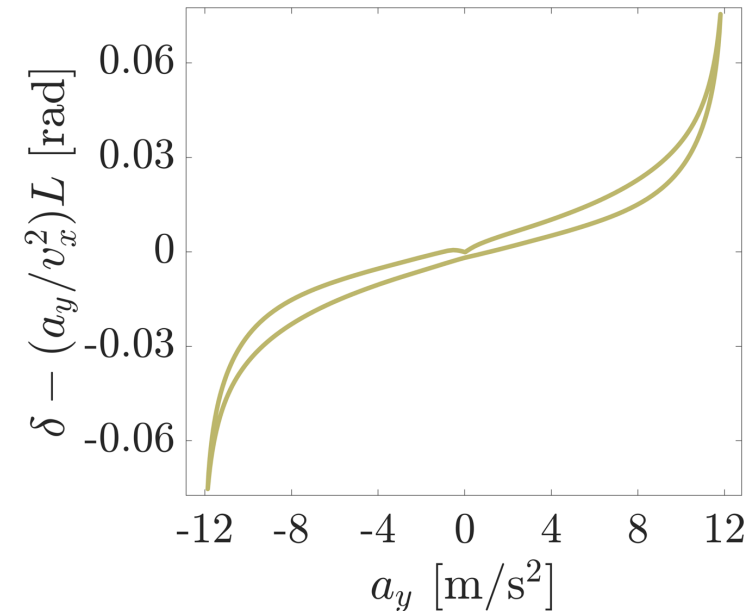
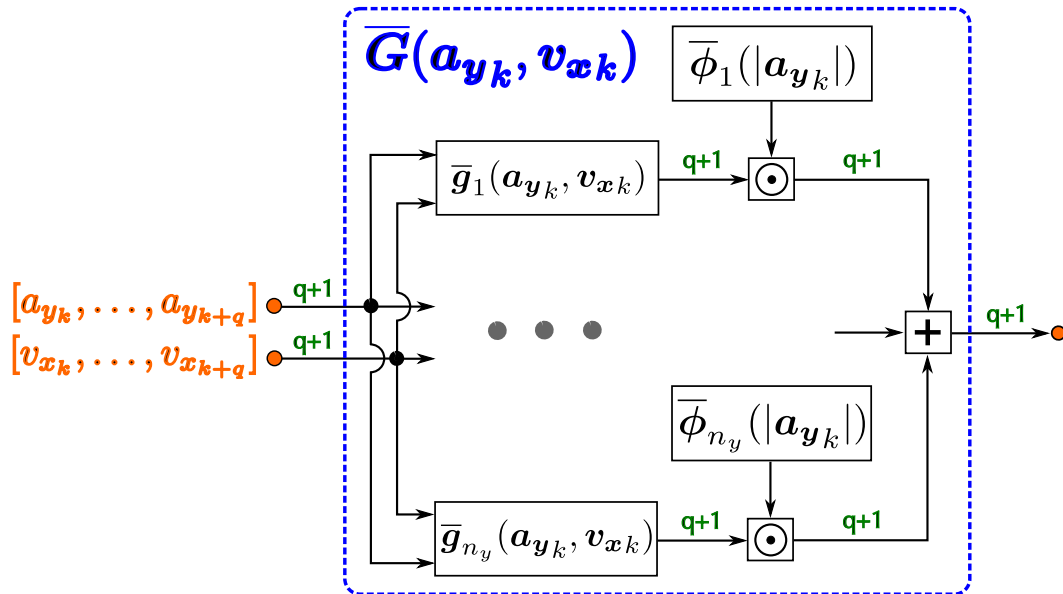
- **Physics-encoded** design: **Handling Diagram**



Steering Control

MS-NN-base: Steady-state pure lateral dynamics

- **Physics-encoded** design: **Handling Diagram**
- Neural compositional modules

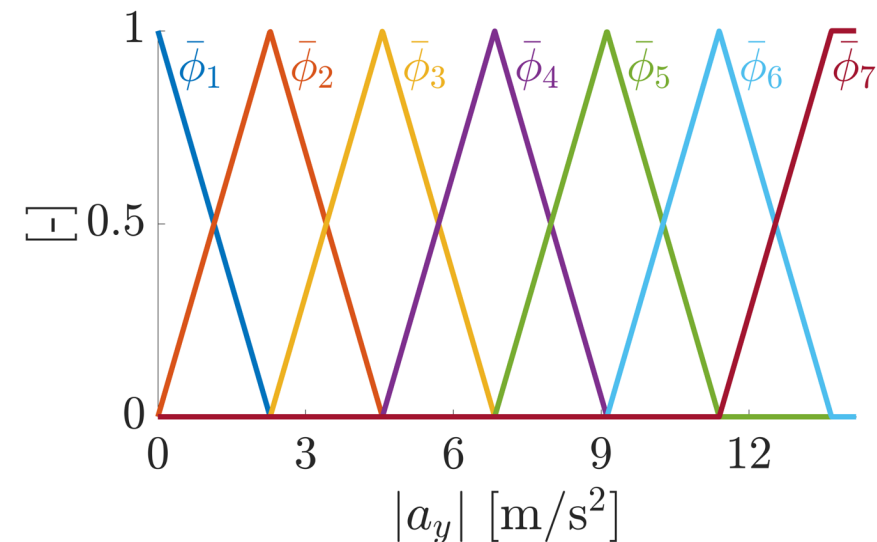
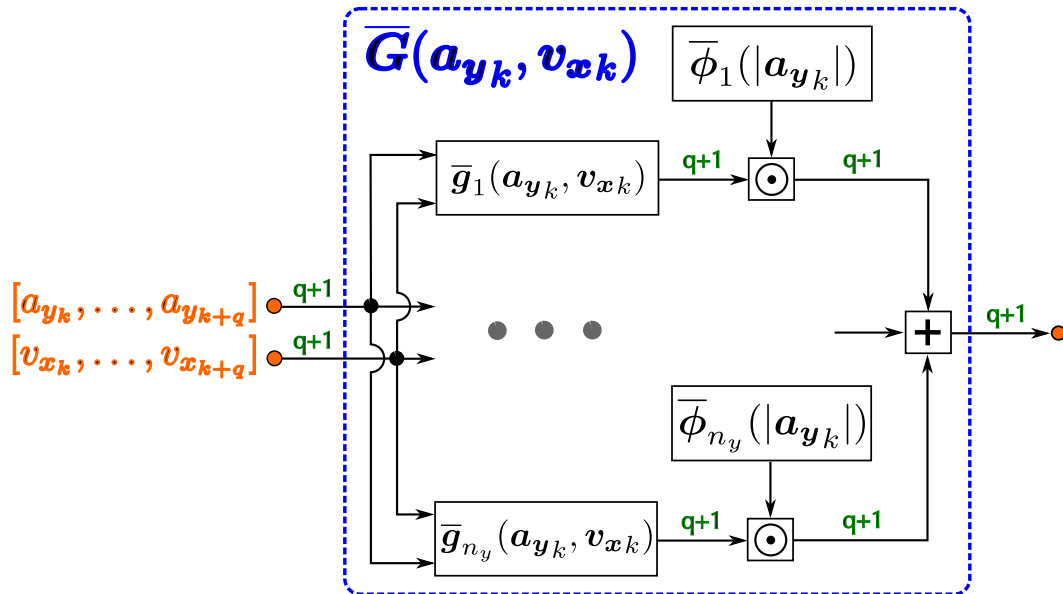


Steering Control

MS-NN-base: Steady-state pure lateral dynamics

- **Physics-encoded** design: **Handling Diagram**
- Neural compositional modules encoding symmetry and a_y dependency

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

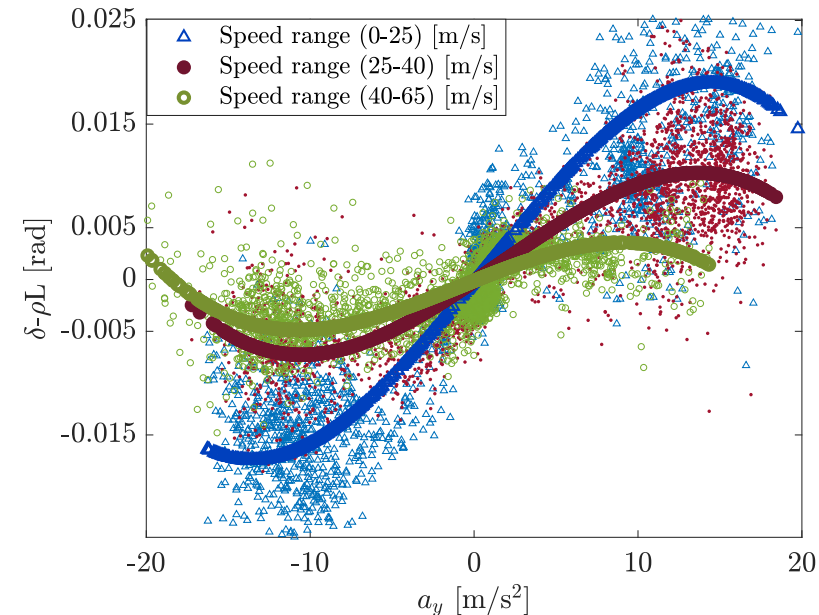
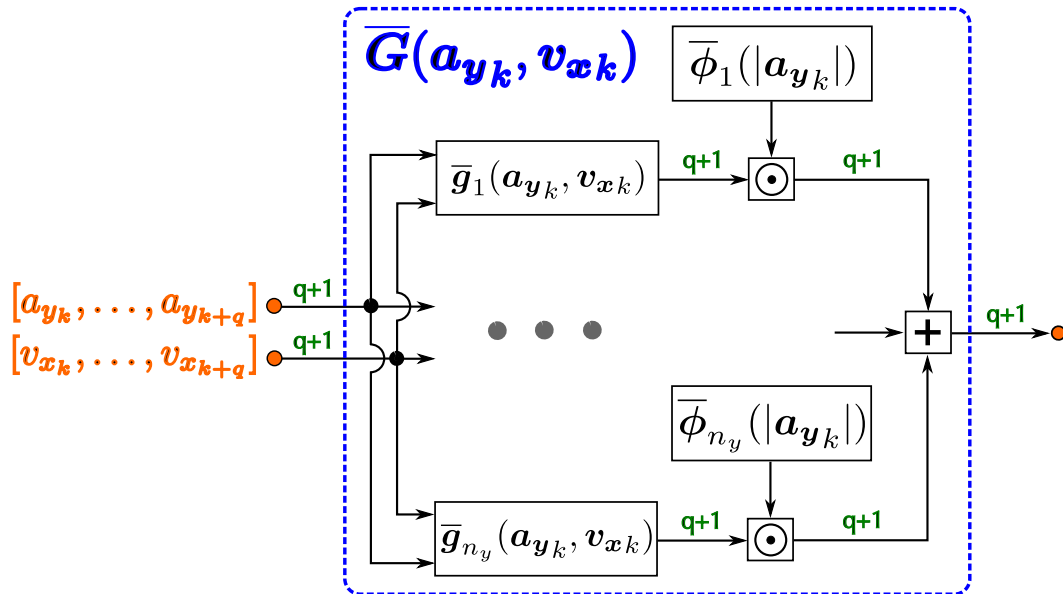


Steering Control

MS-NN-base: Steady-state pure lateral dynamics

- **Physics-encoded** design: **Handling Diagram**
- Neural compositional modules encoding symmetry and a_y dependency

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

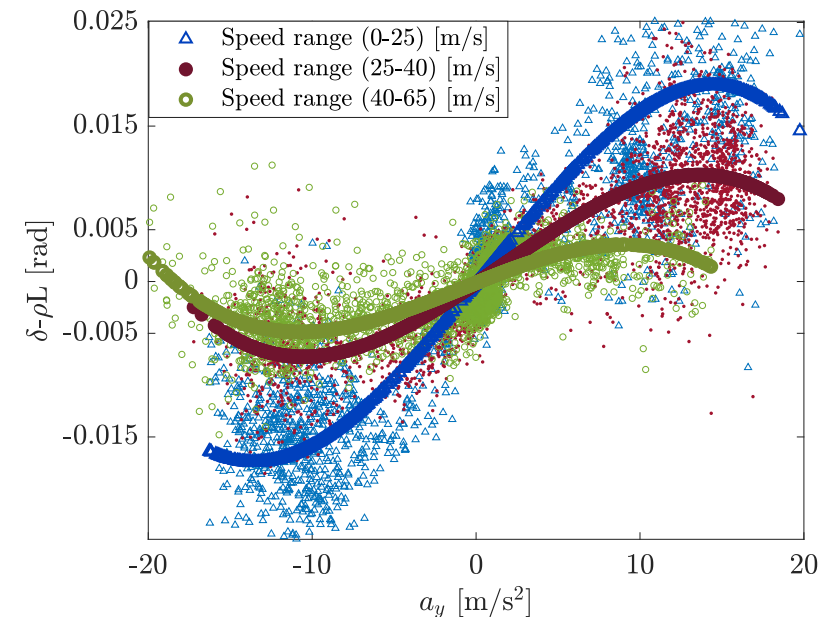
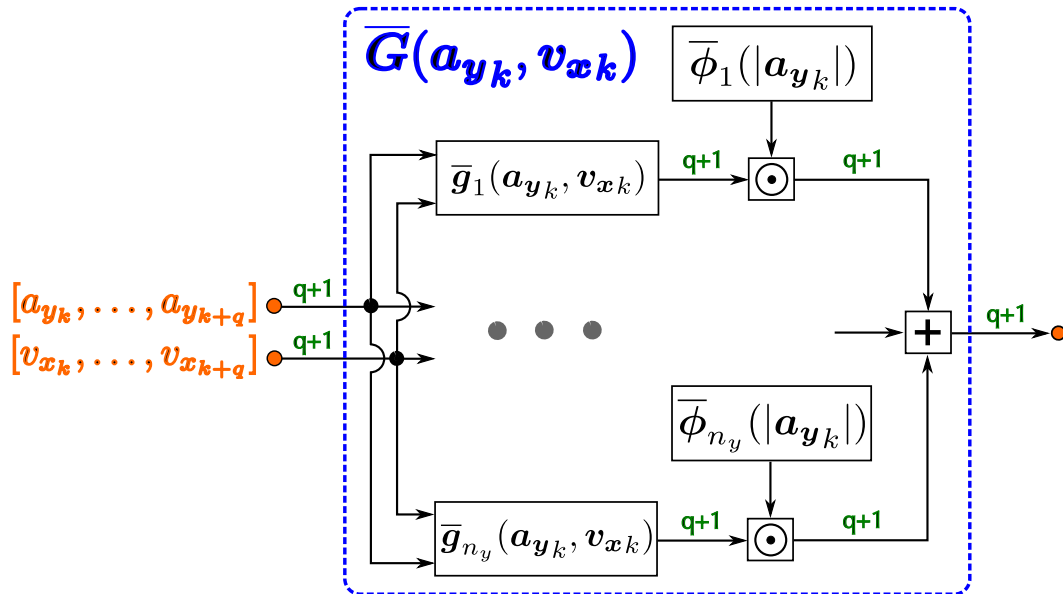


Steering Control

MS-NN-base: Steady-state pure lateral dynamics

- **Physics-encoded** design: **Handling Diagram**
- Neural compositional modules **with v_x dependency**

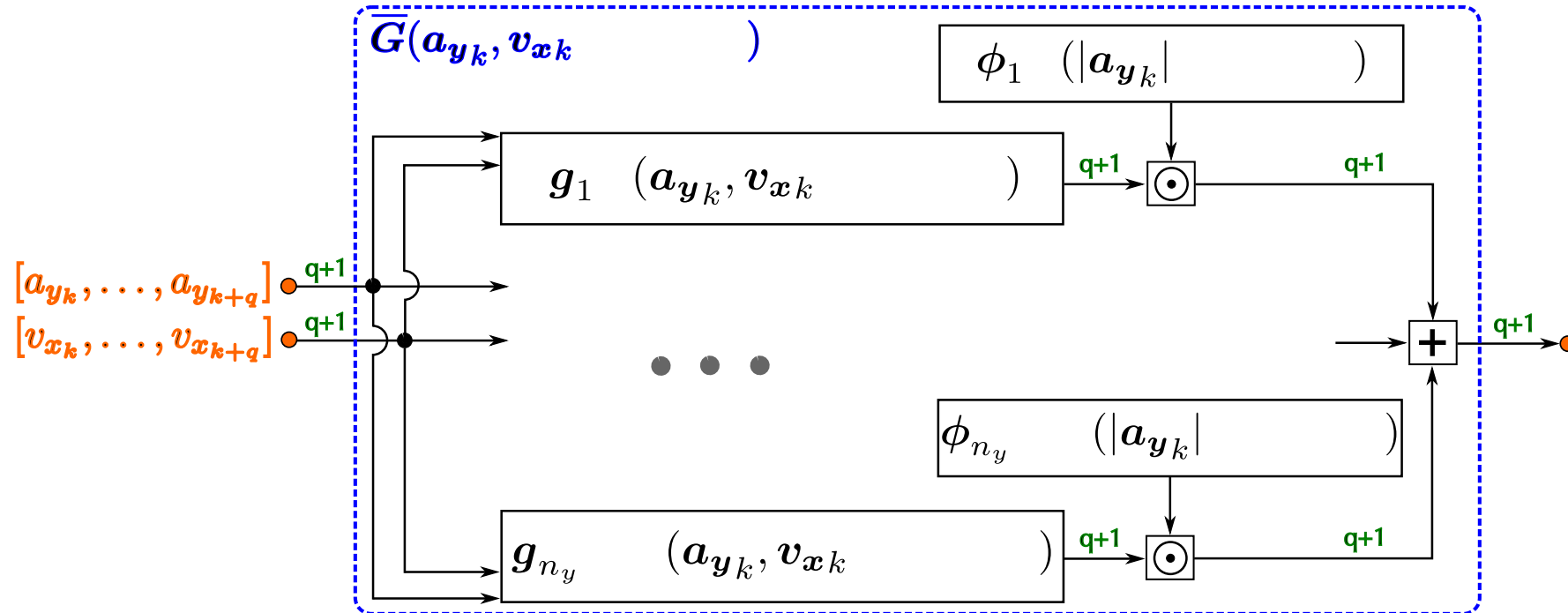
$$\bar{g}_i(a_y, v_x) = \frac{a_y}{v_x^2} L + \text{sign}(a_y) k_{1_i}(v_x) + (a_y - a_{y0_i} \text{sign}(a_y)) k_{2_i}(v_x)$$



Steering Control

MS-NN-ext: **Quasi**-steady-state **combined** lateral dynamics

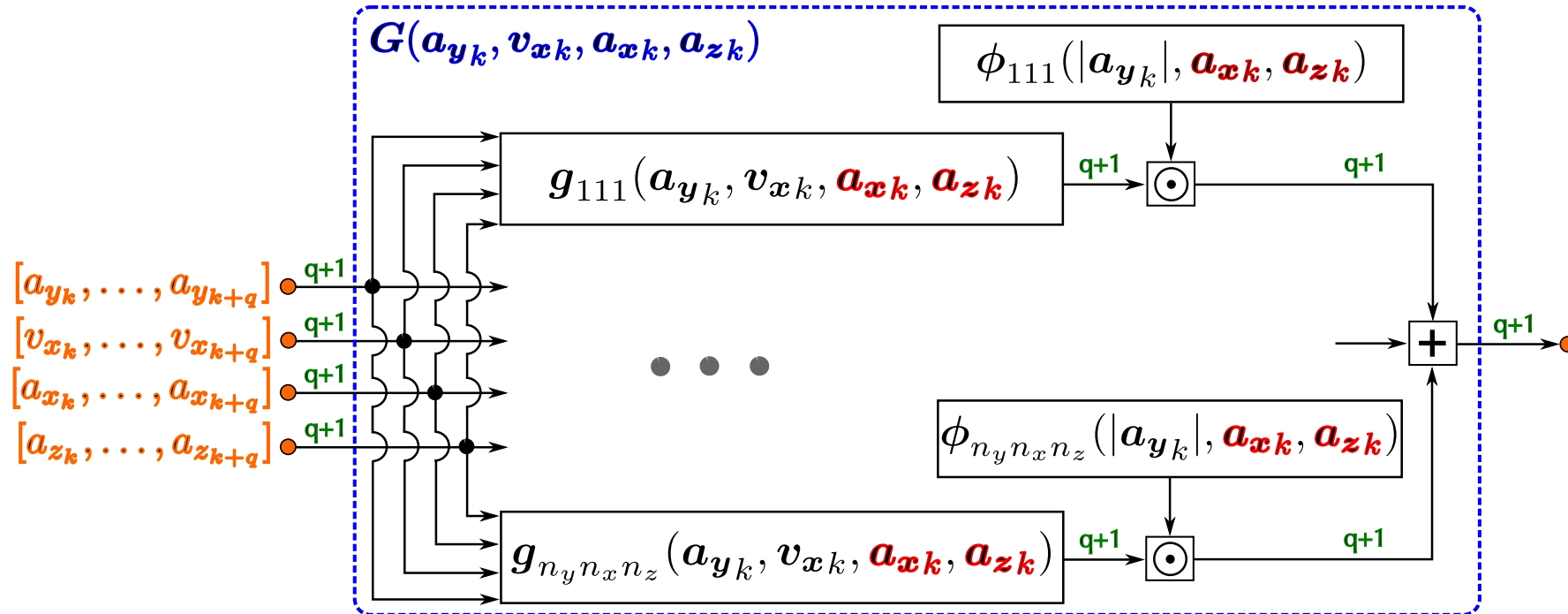
- **Neural compositional modules** in overlapping regions of a_y



Steering Control

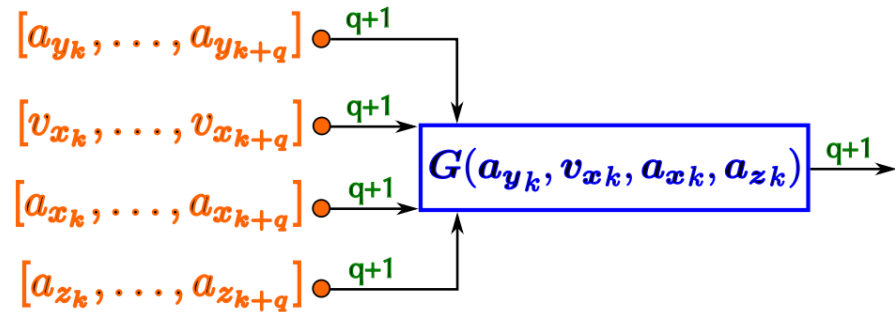
MS-NN-ext: **Quasi**-steady-state **combined** lateral dynamics

- **Neural compositional modules** in overlapping regions of a_y , a_x , a_z
- $g_{ijk}(\cdot) \rightarrow$ Local approximation of a double-track model around $\{a_{y_{0i}}, a_{x_{0j}}, a_{z_{0k}}\}$



Steering Control

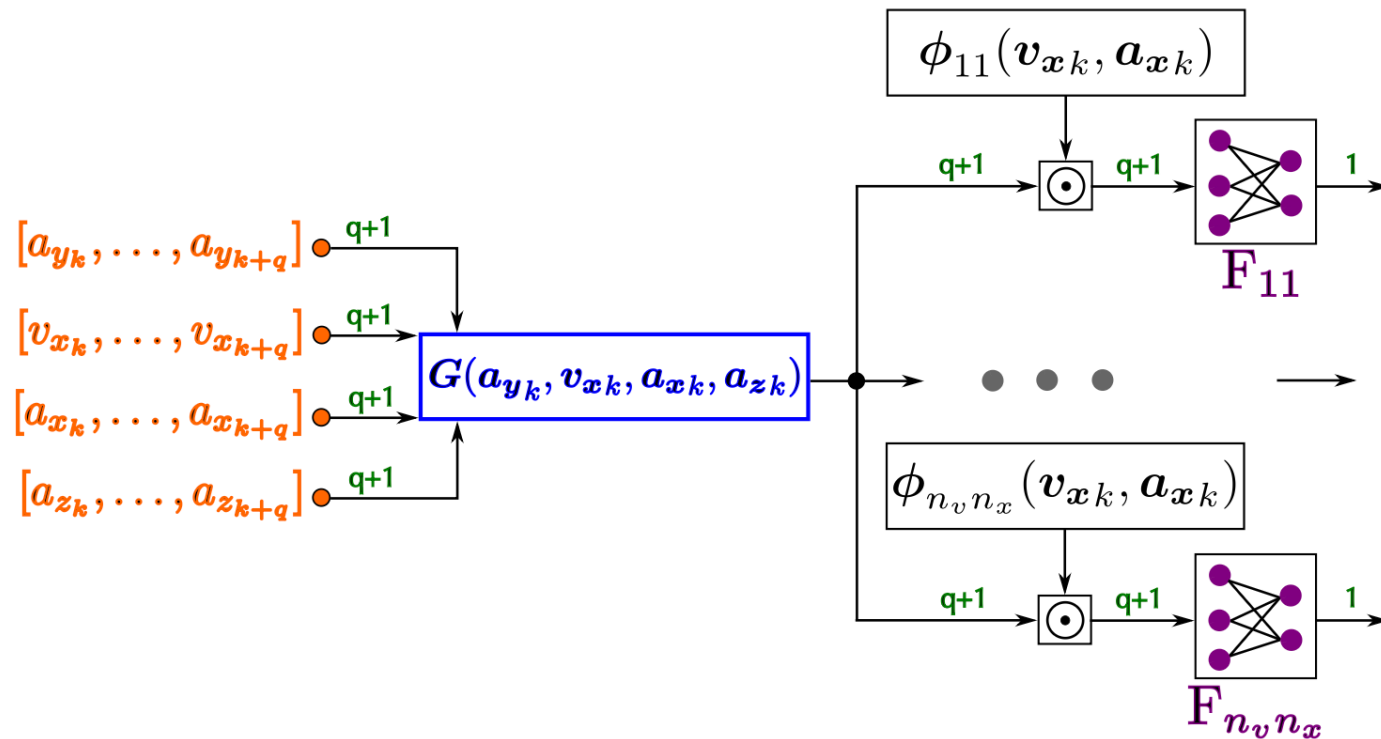
MS-NN-ext: **Transient** combined lateral dynamics



Steering Control

MS-NN-ext: **Transient** combined lateral dynamics

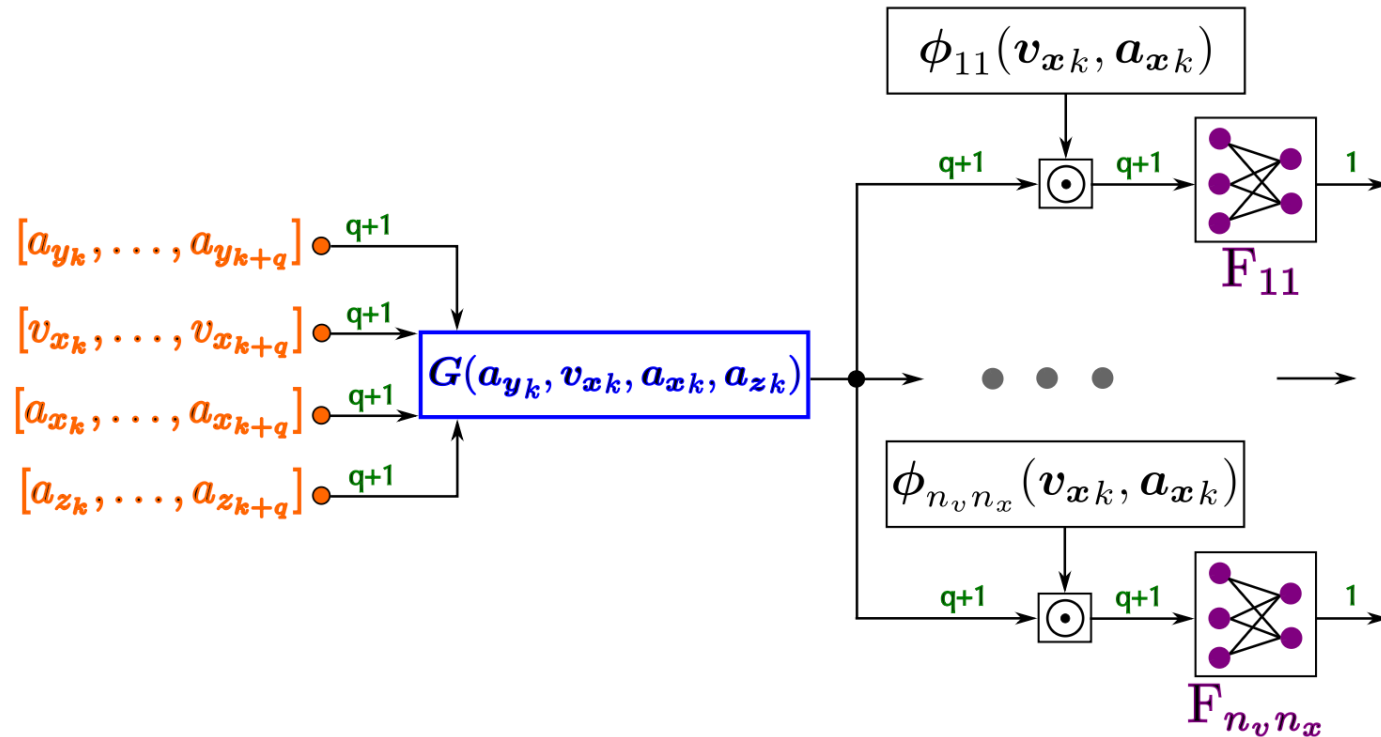
- **Physics-encoded** design: transient response changes with v_x , a_x



Steering Control

MS-NN-ext: **Transient** combined lateral dynamics

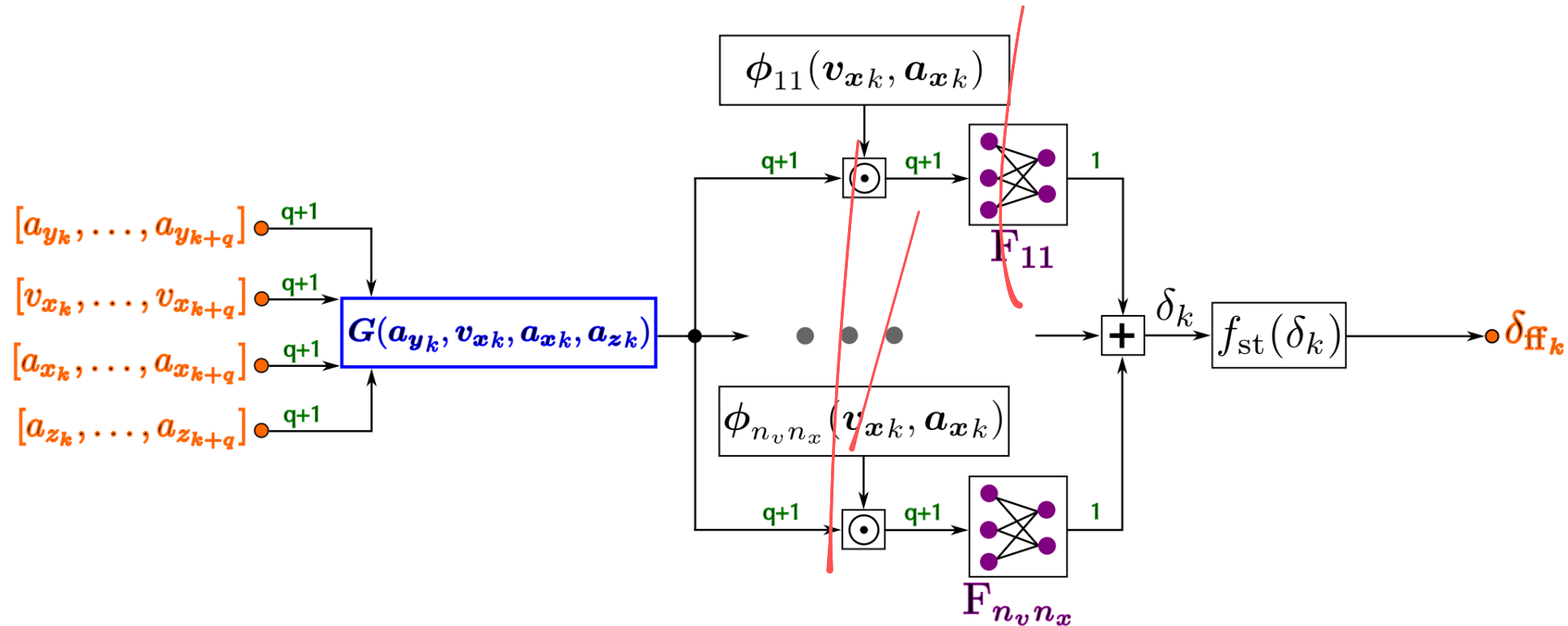
- **Physics-encoded** design: transient response changes with v_x , a_x
- Local fully-connected layers in overlapping regions of v_x , a_x



Steering Control

MS-NN-ext: **Transient** combined lateral dynamics

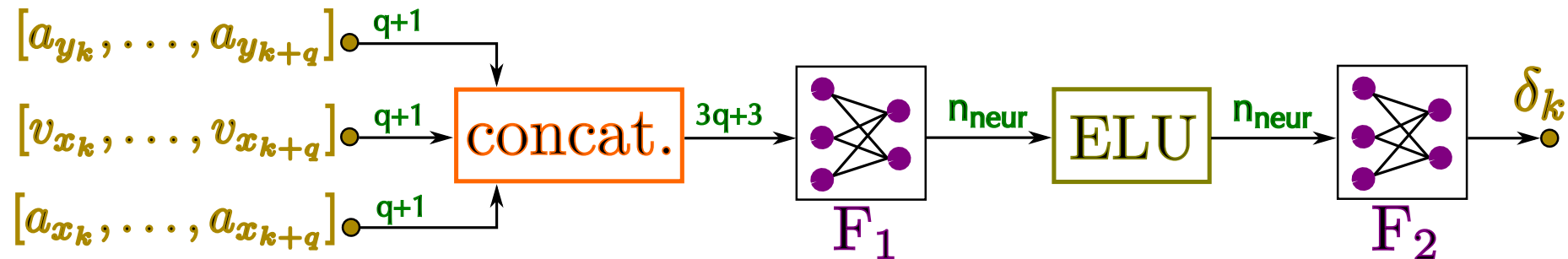
- **Physics-encoded** design: transient response changes with v_x , a_x
- Local fully-connected layers in overlapping regions of v_x , a_x



Steering Control - Results

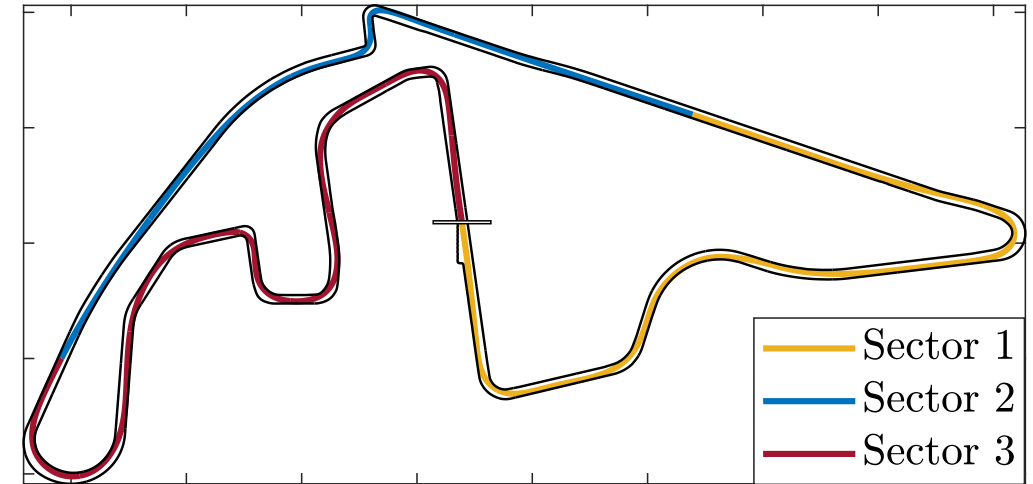
Benchmarks

- A general-purpose NN (**G-NN**)
- A baseline MS-NN from Piccinini et al., 2023 (**MS-NN-base**)
- Heuristic-based steering controller of TUM-A2RL (**A2RL-control**)



$$\delta_k = \delta_{\text{kin}}(a_{y_k}, v_{x_k}) + \delta_{\text{us}}(a_{y_k}) + \delta_{\text{ax}}(a_{y_k}, a_{x_k}) + \delta_{\text{off}}$$

Steering Control - Results

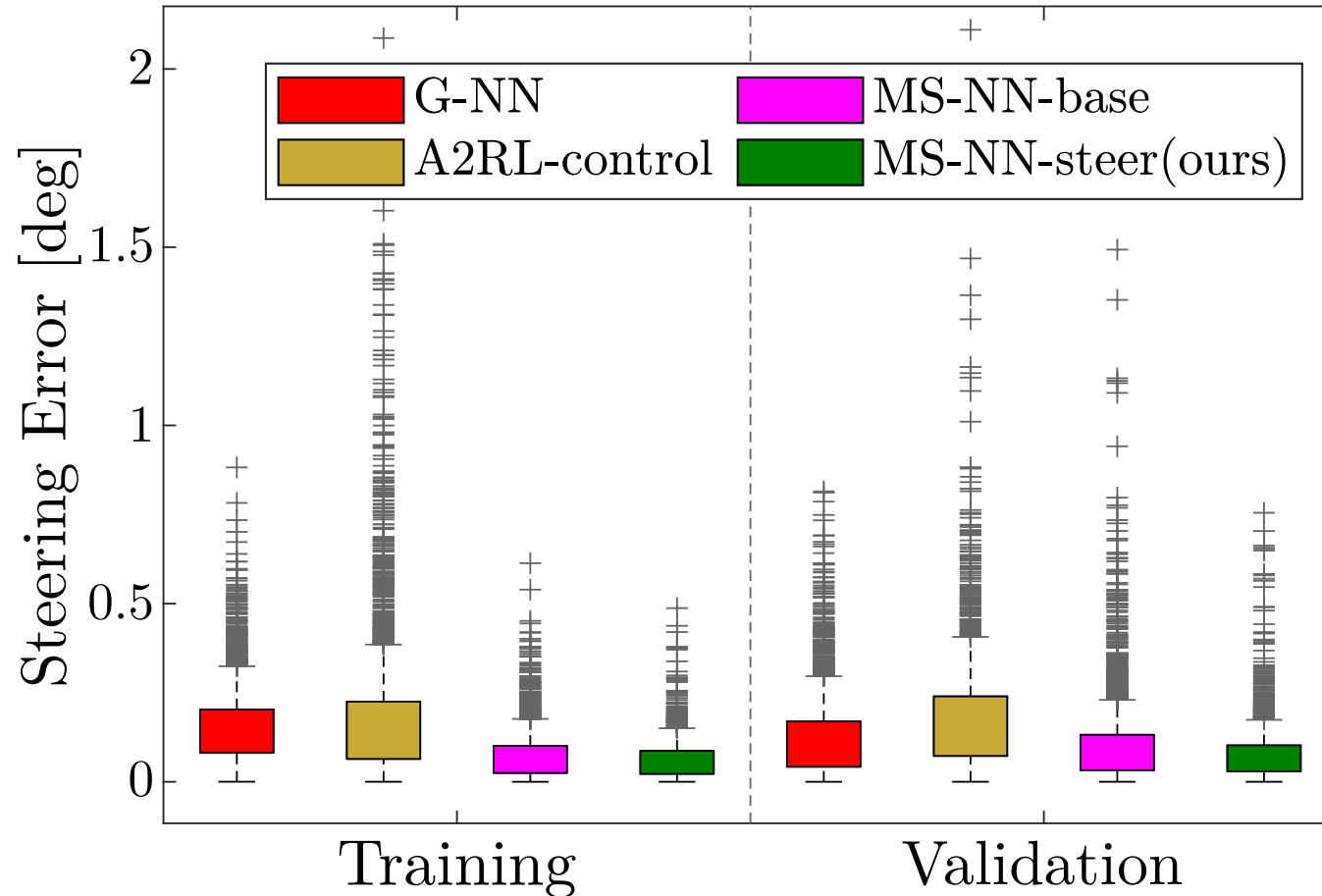


Training with **one lap** (or less!) of real-world data

	Training sets			Validation set
	Small	Medium	Large	
Track sectors	3	1+3	1+2+3	1+2+3
Lap	1	1	1	2

Steering Control - Results

«Large» training dataset



Better **accuracy** &
generalization

Steering Control - Results

Smaller training datasets

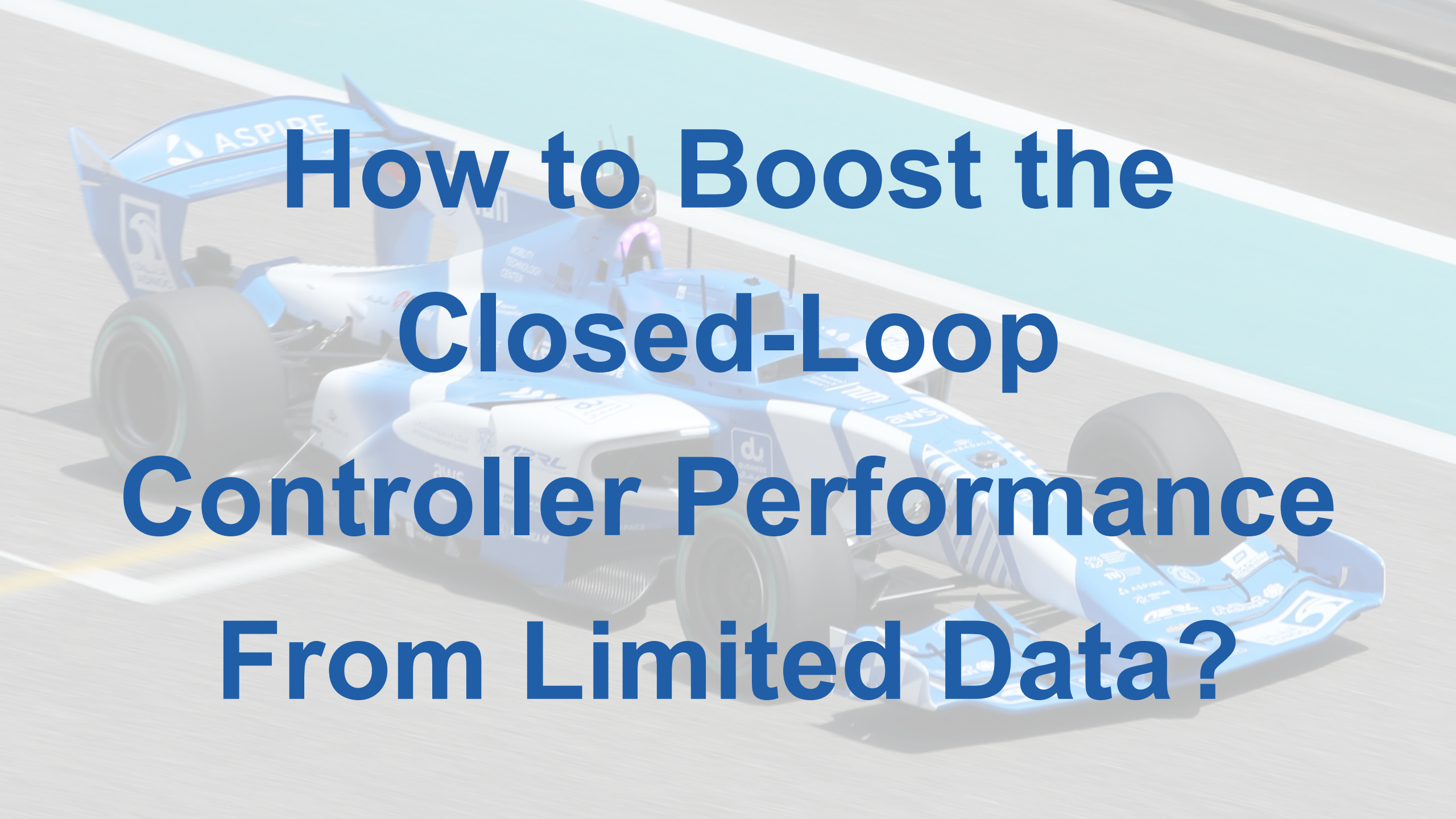
		Medium Train. Set		Small Train. Set	
	Metrics	Train.	Valid.	Train.	Valid.
G-NN	RMSE [deg]	0.160	0.229	0.132	0.327
	FVU [-]	0.0059	0.0150	0.0025	0.0792
MS-NN-base	RMSE [deg]	0.079	0.097	0.069	0.115
	FVU [-]	0.0011	0.0021	0.0006	0.0026
MS-NN-steer (ours)	RMSE [deg]	0.068	0.091	0.063	0.097
	FVU [-]	0.0008	0.0018	0.0006	0.0020

Better **generalization from small training datasets**
thanks to the **physics-based priors**

Steering Control

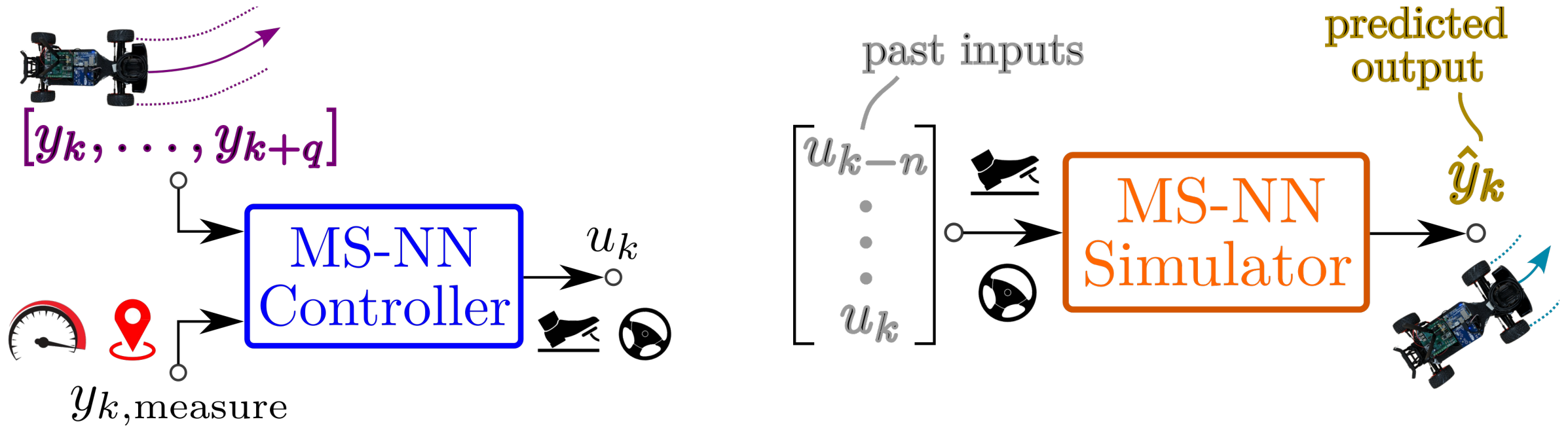
Implementing this in **NModely** is very simple: you find it [here](#)!

The logo for NModely, featuring a stylized lowercase 'm' with a blue-to-black gradient, followed by the word 'odely' in a black sans-serif font.



How to Boost the Closed-Loop Controller Performance From Limited Data?

Boosting Controller Performance from Limited Data



Can we let the **controller** learn to control the **simulator** in a *closed loop*?

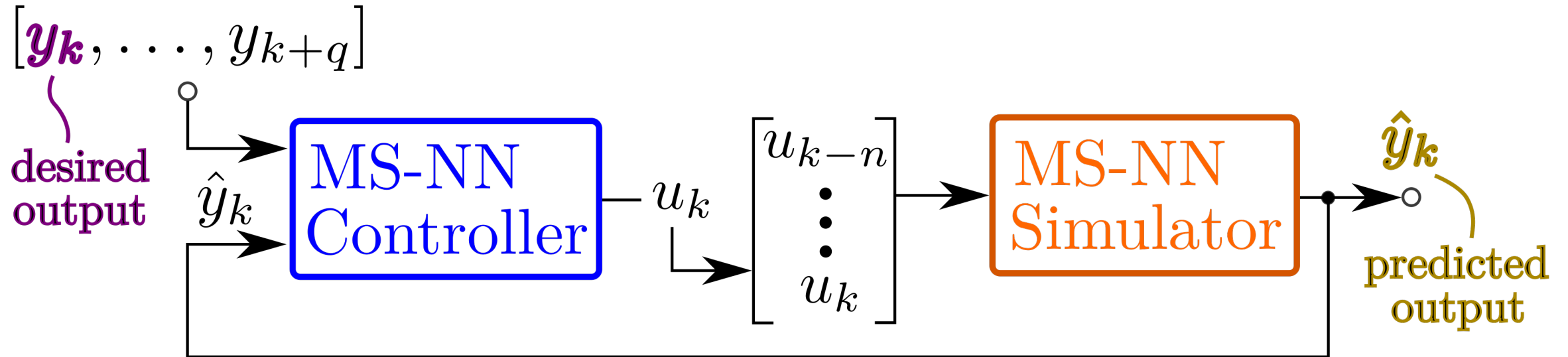
Boosting Controller Performance from Limited Data

1. Train the **MS-NN Simulator** with supervised learning from a collected dataset



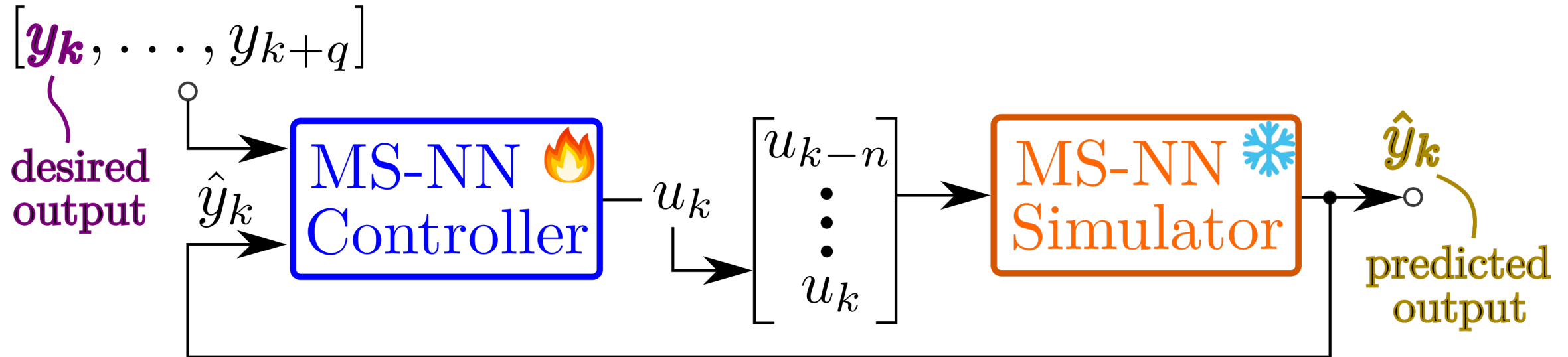
Boosting Controller Performance from Limited Data

2. Couple the **MS-NN Simulator** (trained) with the **MS-NN Controller** (untrained)



Boosting Controller Performance from Limited Data

3. Train the **MS-NN Controller** in *closed-loop* (frozen simulator)



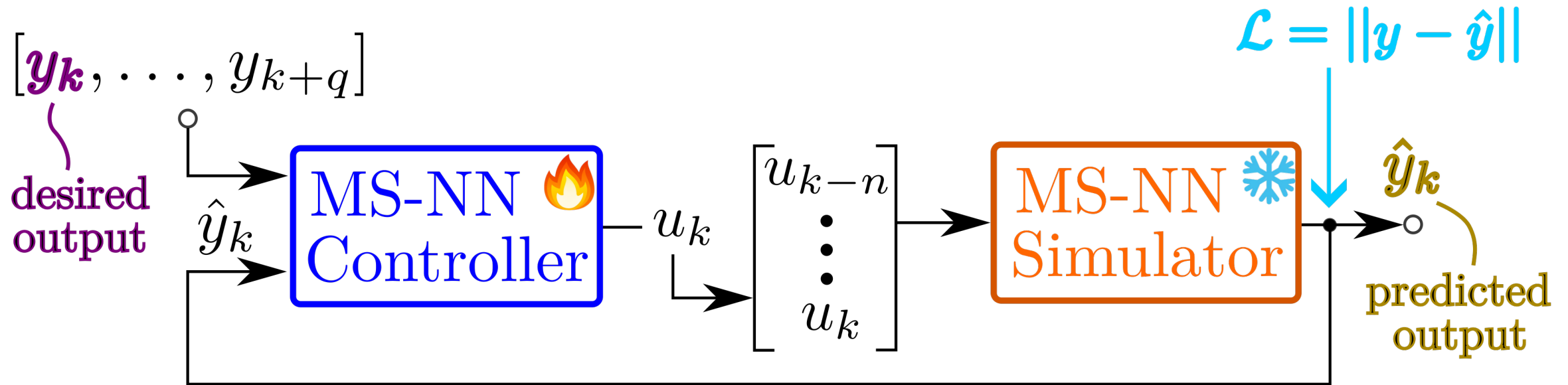
How to do it? Any ideas?

Common approach: **RL** → But it requires **millions of interactions**

→ hard to be done online / on the physical system → How to improve?

Boosting Controller Performance from Limited Data

3. Train the **MS-NN Controller** in *closed-loop* (frozen simulator)

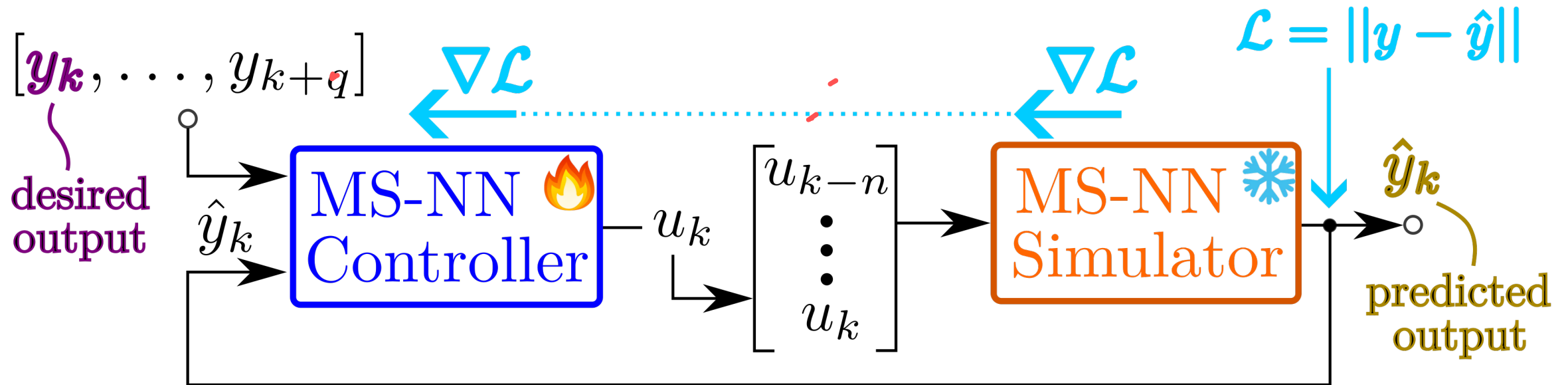


Unsupervised loss $\mathcal{L}$

→ the controller can be **trained end-to-end (E2E)** without ground-truth data!

Boosting Controller Performance from Limited Data

3. Train the **MS-NN Controller** in *closed-loop* (frozen simulator)



Differentiable Closed-loop Controller Training

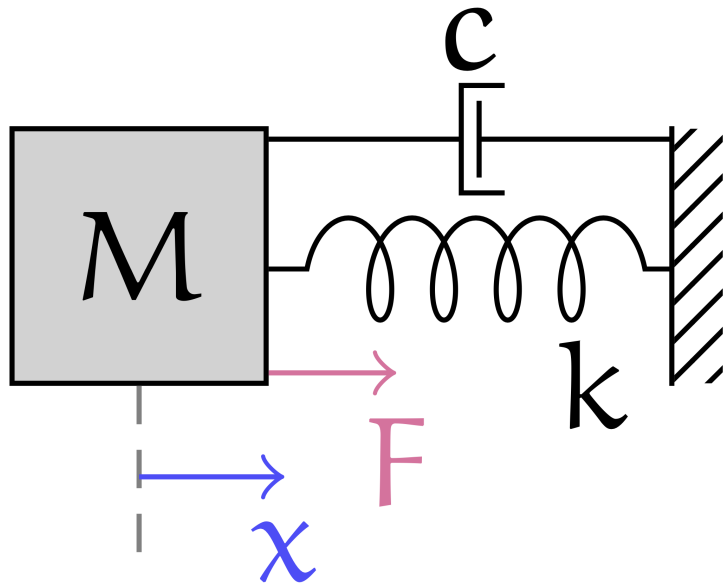
$$\theta_{k+1} = \theta_k - \eta \nabla \mathcal{L}_k \implies \text{Gradient-based (e.g., ADAM)}$$

Enables data-free **training & adaptation**

Boosting Controller Performance from Limited Data

An example of this in **NNodey** with a **mass-spring-damper** system is [here](#)!

Physical System



$$M\ddot{x} + c\dot{x} + kx = F$$

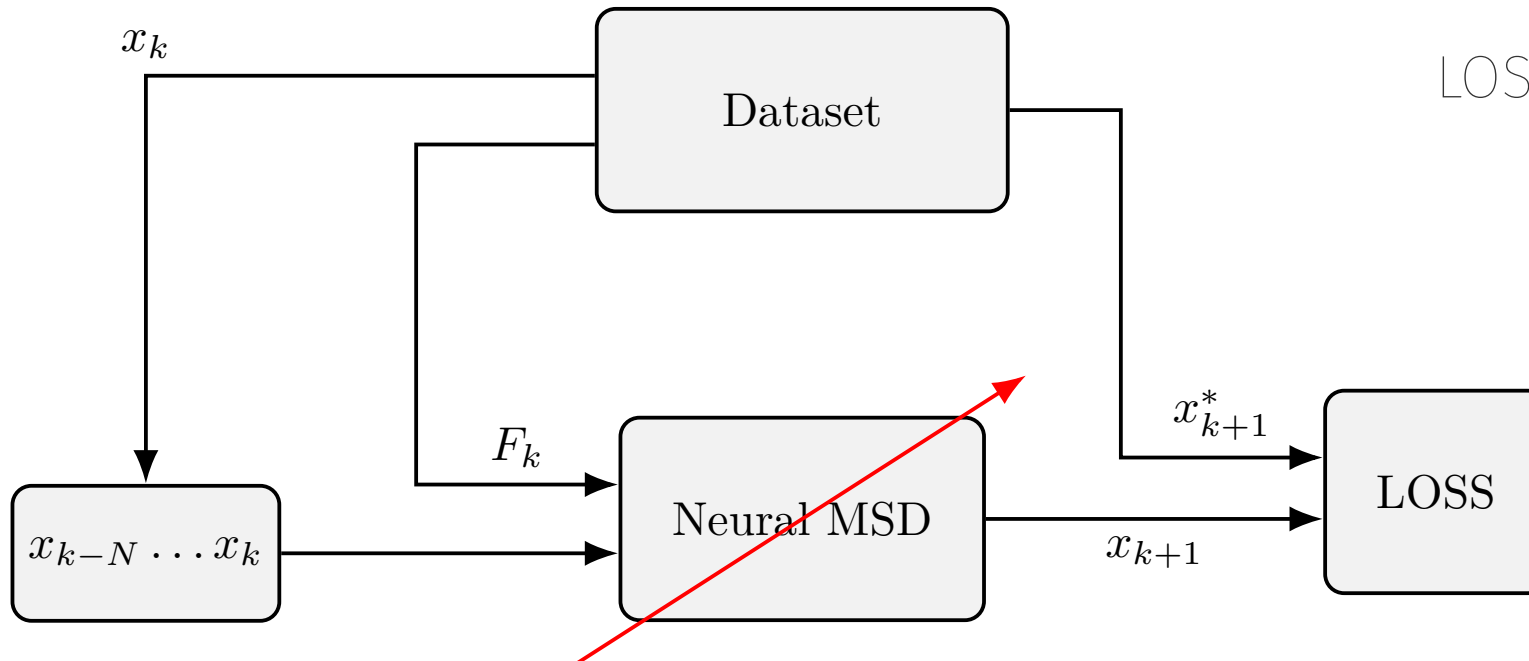
Proposed MS-NN Design

$$x_{k+1} = \underbrace{\sum_{k=0}^{N-1} x[-k] \cdot h_x[(N-1)-k]}_{\text{FIR}(\bar{x})} \underbrace{\frac{c\dot{x}+kx}{M}}_{\text{FIR}(F)} + \underbrace{\frac{F}{M}}_{\text{FIR}(F)} \underbrace{F[0] \cdot h_F}_{\text{FIR}(F)}$$

Boosting Controller Performance from Limited Data

An example of this in **NNode** with a **mass-spring-damper** system is [here](#)!

Step 1: dynamics learning (direct model)



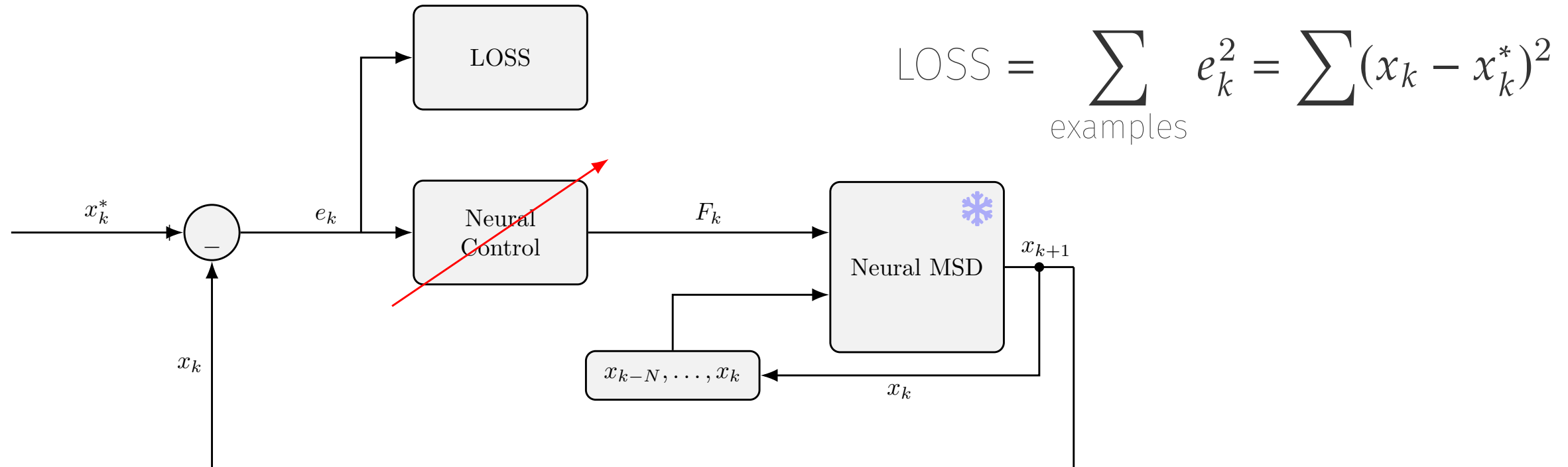
$$\text{LOSS} = \sum_{\text{examples}} e_k^2 = \sum (x_k - x_k^*)^2$$

Boosting Controller Performance from Limited Data

An example of this in **NNode** with a **mass-spring-damper** system is [here](#)!

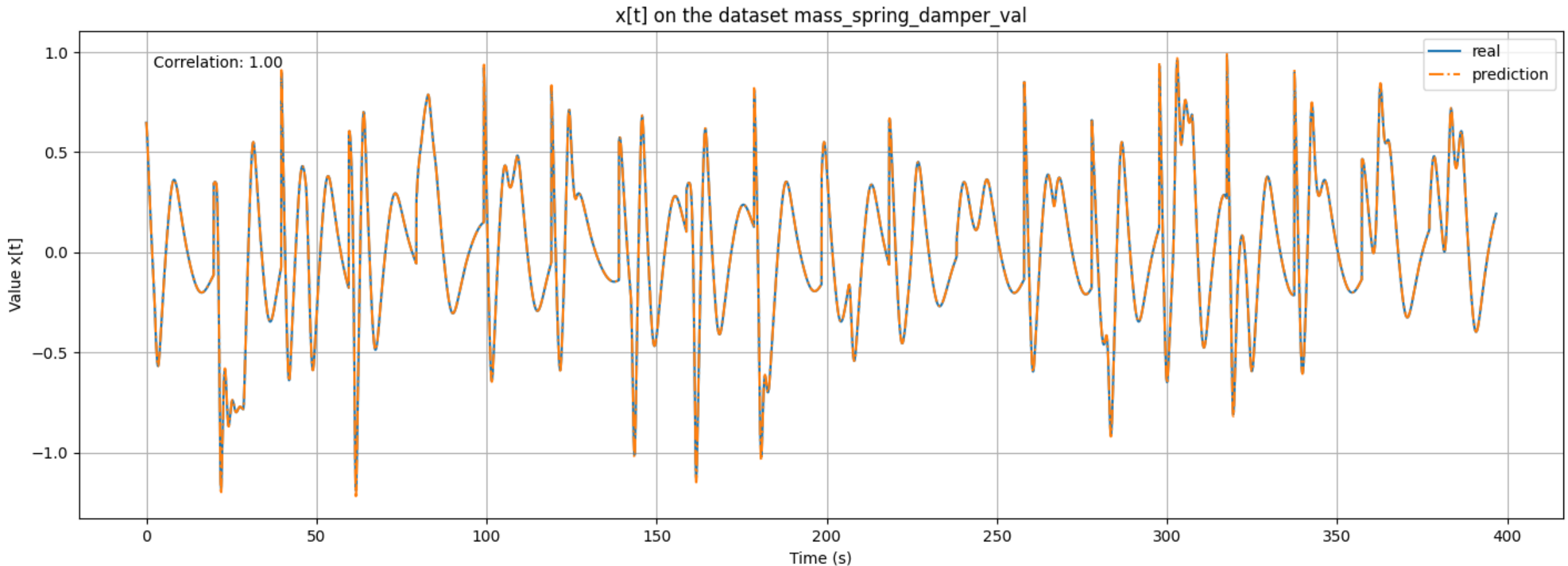
Step 2: controller learning (inverse model) through the pre-trained direct model

➔ **Unsupervised learning**



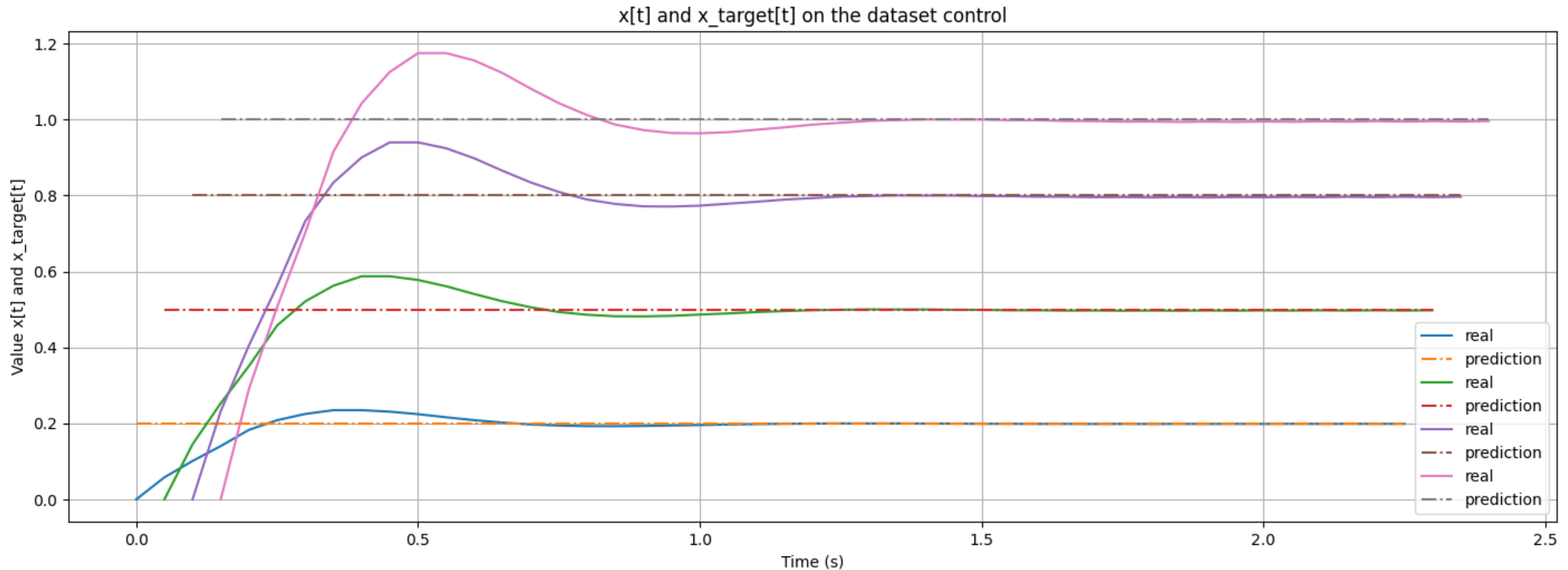
Boosting Controller Performance from Limited Data

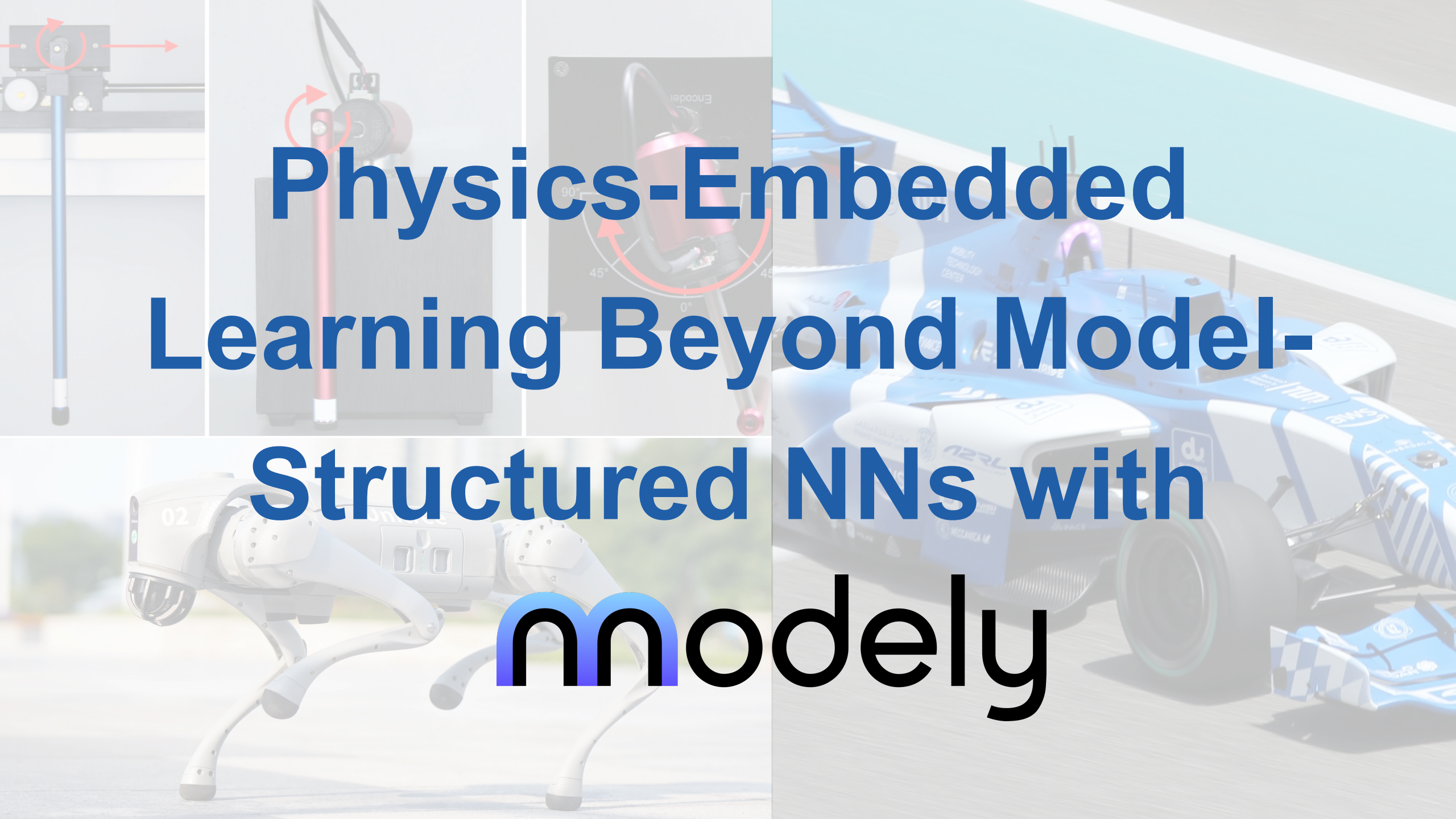
An example of this in **NNode** with a **mass-spring-damper** system is [here](#)!



Boosting Controller Performance from Limited Data

An example of this in **NNode** with a **mass-spring-damper** system is [here!](#)





Physics-Embedded Learning Beyond Model- Structured NNs with modely

Other Examples with NNodey: not only Model-Structured NNs!

- *Nonlinear function fitting*
- *Sobolev learning*
- *Physics informed neural network*

modely

Questions & Discussion

