Neural Solvers for Fast and Accurate Numerical Optimal Control

Federico Berto[1], Stefano Massaroli[2]
Michael Poli[3], Jinkyoo Park[1]

[1]KAIST, [2]The University of Tokyo, [3]Stanford University

10th International Conference on Learning Representations
ICLR 2022

Optimal Control Problem

Optimal Control Problem: find controller parameters θ to minimize cost

$$\begin{aligned} \min_{\theta} \quad & \mathbb{E}_{x_0 \sim \rho_0(x_0)} \left[J(x_0, u_\theta(t)) \right] \\ \text{subject to} \quad & \dot{x}(t) = f(t, x(t), u_\theta(t)) \\ & x(0) = x_0; \;\; t \in \mathcal{T} \end{aligned}$$

with cost functional

$$J = [x^\top(t_f) - x^\star]\mathbf{P}[x(t_f) - x^\star] + \int_{t_0}^{t_f} \left([x^\top(t) - x^\star]\mathbf{Q}[x(t) - x^\star] + {u_\theta}^\top(t)\mathbf{R}u_\theta(t) \right) dt$$

where the weights $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ denote the *terminal cost*, *trajectory cost* and *controller regulator* respectively

Numerical Solution of the Control Problem

Explicit ODE Solvers: iterate to solve

$$x_{k+1} = x_k + \epsilon \psi_\epsilon \left(t_k, x_k, u_k\right)$$

- ODE solvers differ in how the map ψ is designed
- *Higher–order* (explicit) solvers compute ψ iteratively in p steps (p denotes the order of the solver)

Solver Residuals: given nominal solutions Φ of the optimal control problem

$$R_k = \frac{1}{\epsilon^{p+1}} \left[\Phi(x_k, t_k, t_{k+1}) - x_k - \epsilon \psi_\epsilon (t_k, x_k, u_k) \right]$$

Some Key Issues

There are *key issues* to take into consideration when solving the optimal control problem:

- **Accuracy** is essential to obtain correct solutions
- **Speed** is crucial for hard–time constrained applications
- Fast but inaccurate solvers **trade off** solution accuracy for speed

Can we do better?

Yes. Extend the paradigm of *hypersolvers*[a] to controlled systems

[a] Poli et al., *Hypersolvers: Toward Fast Continuous-Depth Models*, NeurIPS 2020

Hypersolvers for Optimal Control

Control input u becomes a key component in hypersolver design

$$x_{k+1} = x_k + \underbrace{\epsilon \psi_\epsilon \left(t_k, x_k, u_k\right)}_{\text{base solver step}} + \epsilon^{p+1} \underbrace{g_\omega \left(t_k, x_k, u_k\right)}_{\text{hypersolver}}$$

Residual Fitting: training of g_ω can be performed by minimizing

$$\ell_\omega = \frac{1}{K} \sum_{k=0}^{K-1} ||R_k - g_\omega \left(t_k, x_k, u_k\right)||_2$$

$\Longrightarrow$ Hypersolvers benefit from **theoretical guarantees** on error bounds!

Hypersolver Training

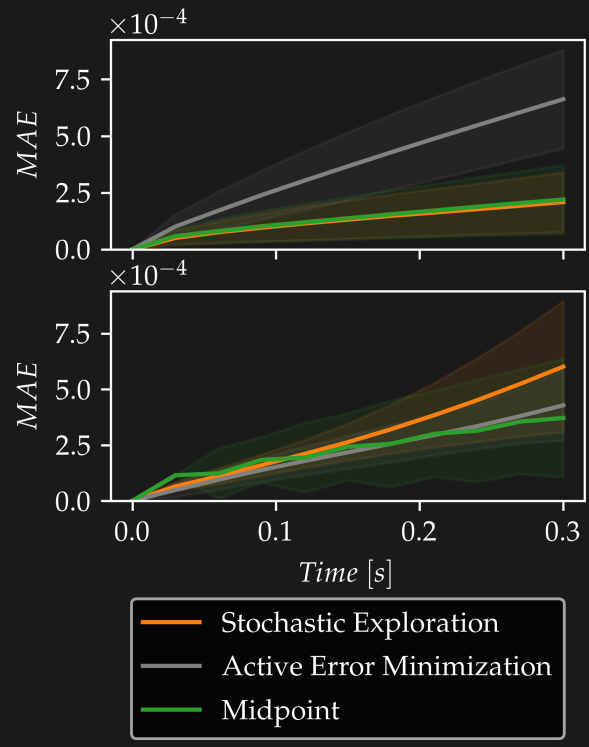
Exploration Strategies: we train hypersolvers on a distribution ξ of states and controllers

- Directly optimize the hypersolver by sampling from the distribution (*stochastic exploration*) thus minimizing the average residual

$$\min_{\omega} \quad \mathbb{E}_{(x,u)\sim\xi(x,u)} \; \ell_\omega(t,x,u)$$

- We can also prioritize exploration in regions with higher residuals (*active error minimization*)

$$\min_{\omega} \quad \max_{u\in\mathcal{U}} \; \mathbb{E}_{(x,u)\sim\xi(x,u)} \; \ell_\omega(t,x,u)$$



Multi–Stage Hypersolvers

What if first order errors in the nominal dynamic model are present?

Multi–Stage Hypersolvers

1. Correct vector field with an *inner stage* approximator:

$$f^\star(t_k, x_k, u_k) = \underbrace{f(t_k, x_k, u_k)}_{\text{partial dynamics}} + \underbrace{h_w(t_k, x_k, u_k)}_{\text{inner stage}}$$

2. Apply *outer stage* hypersolver:

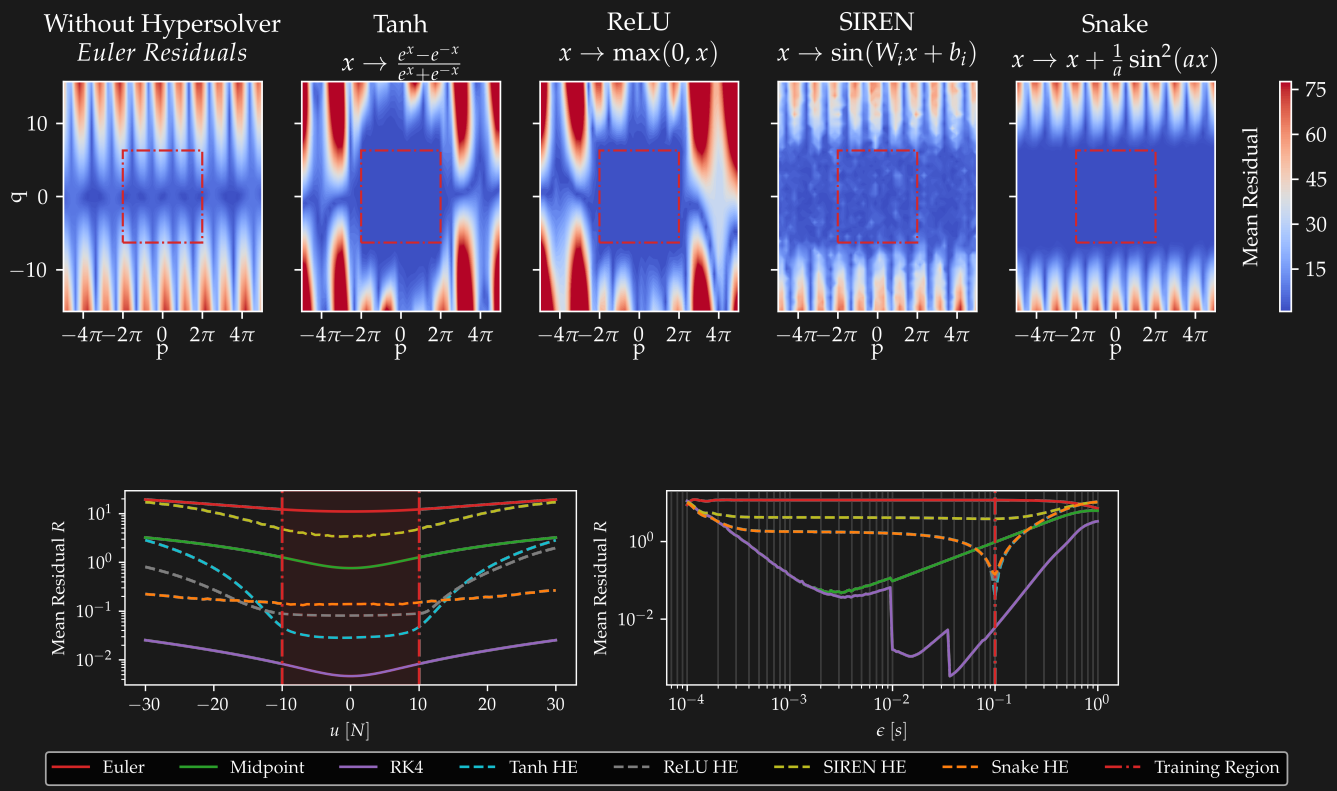
$$x_{k+1} = x_k + \epsilon \psi_\epsilon \left(t_k, x_k, u_k, \underbrace{f^\star(t_k, x_k, u_k)}_{\text{corrected dynamics}} \right) + \epsilon^{p+1} \underbrace{g_\omega(t_k, x_k, u_k)}_{\text{outer stage}}$$

Trajectory Fitting: training of h_w and g_ω can be performed by minimizing

$$\ell_{w,\omega} = \frac{1}{K} \sum_{k=0}^{K-1} \| x(t_{k+1}), x_{k+1} \|_2$$

Architectures

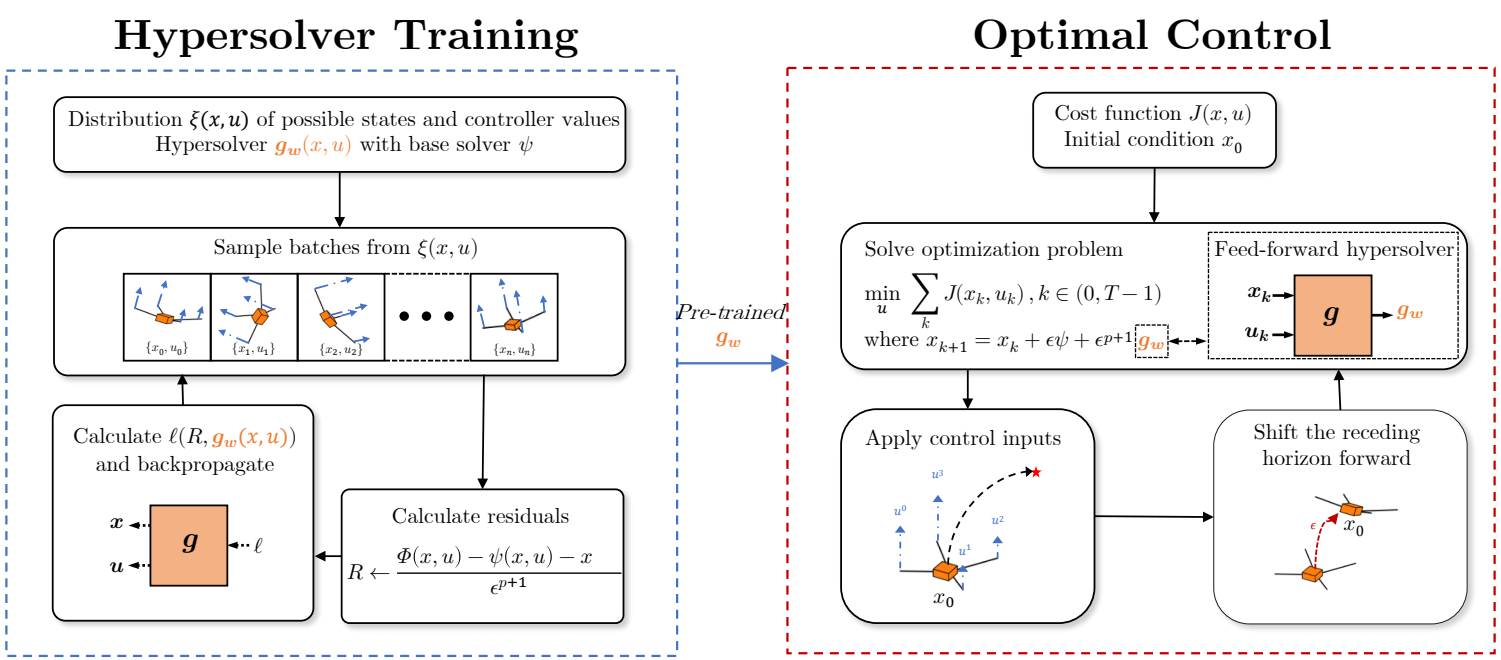
We study different **activation functions** as a core hypersolver design choice



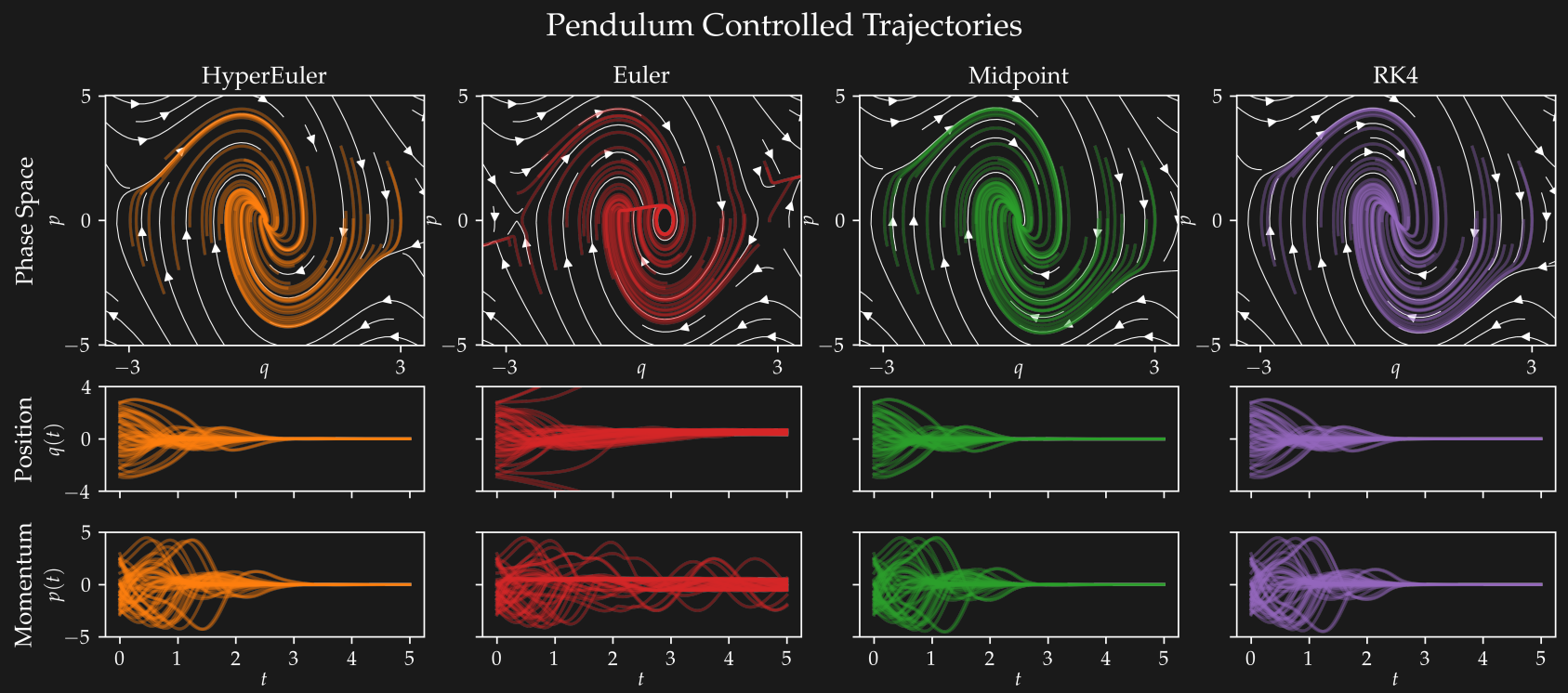
- Different hypersolvers fit the state training region while the ones with *sinusoidal components* also fit better than the base solver outside of it
- Hypersolvers perform better than base solvers even for unseen controllers and timesteps

Blueprint of our Approach

1. **Hypersolver Training**: *offline* optimization of neural approximator
2. **Optimal Control**: *online* optimization of the controller with increased accuracy while introducing a small Hypersolver overhead

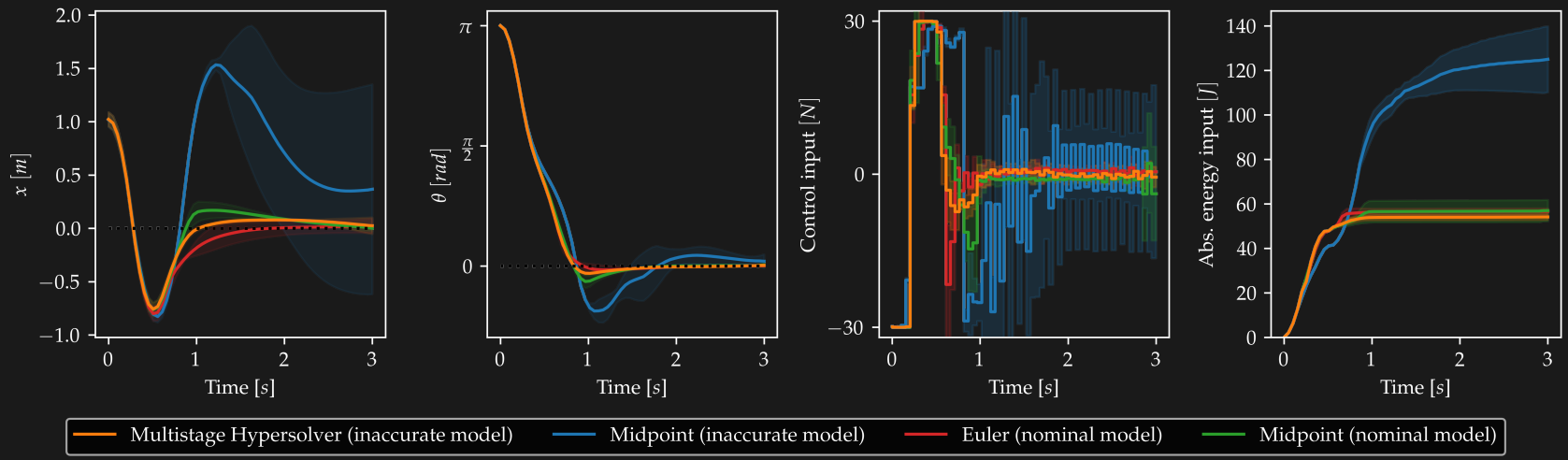


Pendulum Control



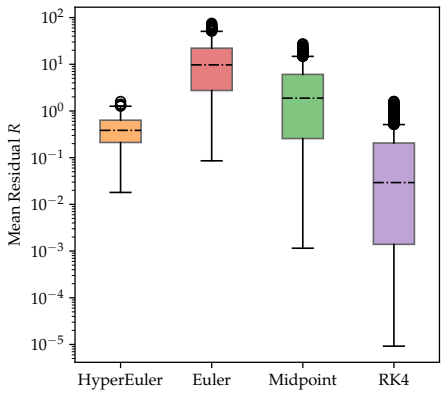
- **Task**: stabilize the pendulum in the vertical position
- Policy comparable to higher–order solvers with almost half the FLOPs

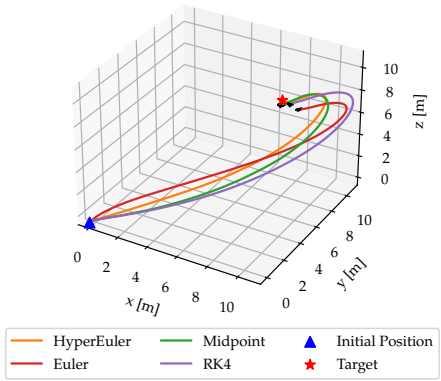
Cartpole Model Predictive Control

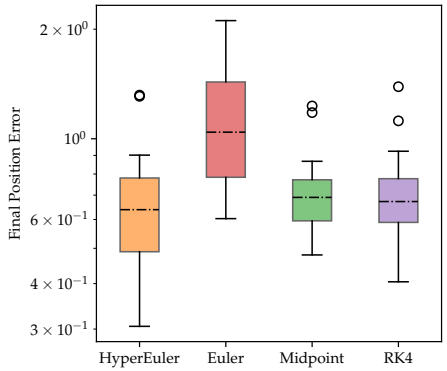


- **Task**: swing-up the cart-and-pole system
- Multi-stage Hypersolver can compensate for the inaccuracy on dynamics and successfully perform the task

Quadcopter Model Predictive Control

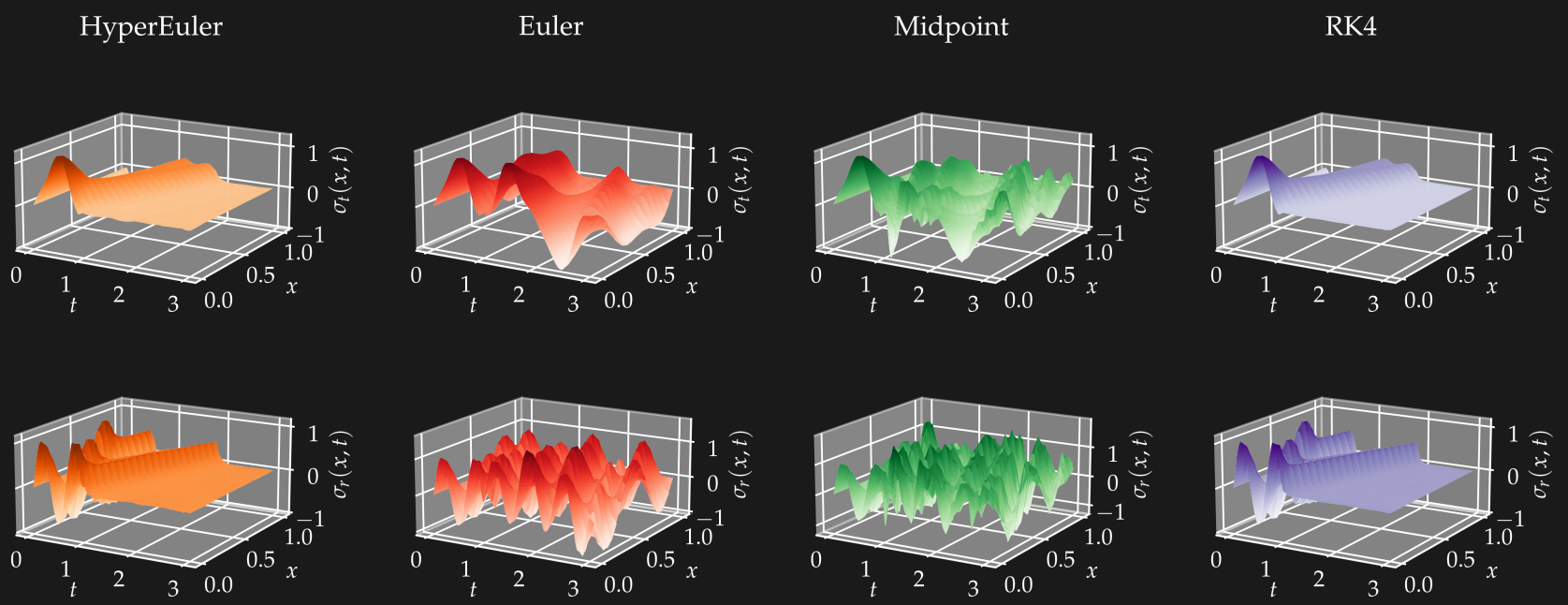






- **Task**: move quadcopter to a target position
- Lower residual and position error than Midpoint with half the NFE

Timoshenko PDE Boundary Control



- **Task**: stabilize the beam in resting position
- We control the beam with HyperEuler with 3× speedup w.r.t. RK4
- Hypersolvers are even more impactful in high–dimensional settings

Summary

Hypersolvers for Optimal Control

Neural solvers for numerical optimal control

- Accuracy improvement with reduced computational cost
- Better downstream control policy
- Even more crucial for high–dimensional control

Some Open Questions:

- How well do hypersolvers perform outside of simulation?
- Can we generalize to different systems?
- What is the most efficient *implicit representation* of hypersolver models?

Thank You

OpenReview: https://openreview.net/forum?id=m8bypnj7Yl5
Github: https://github.com/DiffEqML/diffeqml-research/tree/master/hypersolvers-control

DiffEqML: Open research community working at the intersection between machine learning, differential equations and control. Feel free to join our Slack channel!