Neural Solvers for Fast and Accurate Numerical Optimal Control

Anonymous Author(s)
Affiliation
Address
email

Abstract

Synthesizing optimal controllers for dynamical systems in practice involves solving real–time optimization problems with hard time constraints. These constraints restrict the class of numerical methods that can be applied; indeed, computationally expensive but accurate numerical routines often have to be replaced with fast and inaccurate methods, trading inference time for worse theoretical guarantees on solution accuracy. This paper proposes a novel methodology to accelerate numerical optimization of optimal control policies via *hypersolvers*, hybrids of a base solver and a neural network. In particular, we apply low–order explicit numerical methods for the *ordinary differential equation* (ODE) associated to the numerical optimal control problem, augmented with an additional parametric approximator trained to reduce local truncation errors introduced by the base solver. Given a target system to control, we first pre-train hypersolvers to approximate base solver residuals by sampling plausible control inputs. Then, we use the trained hypersolver to obtain fast and accurate solutions of the target system during optimization of the controller. The performance of our approach is evaluated in direct and model predictive optimal control settings, where we show consistent Pareto improvements in terms of solution accuracy and control performance.

1 Introduction

Emerging applications of optimal control to complex, high–dimensional systems require the implementation of computationally expensive numerical methods for *ordinary differential equations* (ODEs) (Wanner & Hairer, 1996; Butcher, 1997). Here, real–time and hardware constraints render certain numerical methods not applicable, forcing the application of computationally cheaper but otherwise less accurate algorithms. While the paradigm of optimal control has successfully been applied in various domains (Vadali et al., 1999; Lewis et al., 2012; Zhang et al., 2016), these computation budget constraints are still a great challenge (Ross & Fahroo, 2006; Baotić et al., 2008). Furthermore, lower–order explicit ODE solvers are known to be brittle in the presence of stiff dynamical systems (Wanner & Hairer, 1996), resulting in numerical issues and unreliable controllers.

This work alleviates these limitations by detailing a systematic procedure for the *offline* optimization and subsequence *online* application of ODE solvers equipped with a neural network approximator of higher-order local truncation residuals. These hybrid solvers, known as *hypersolvers* (Poli et al., 2020a), employ a low–order, computationally cheaper solver while preserving theoretical local and global solution accuracy guarantees of traditional higher order methods. We extend the range of applicability of hypersolvers to controlled dynamical systems, investigating and developing several strategies to enable their application during online solution of the optimal control problem without *finetuning*. Namely, we consider hypersolvers whose neural network residual approximators are *a priori* optimized on an uncontrolled target system and then applied to significantly speed up iterated

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

simulations of the system later required by the control policy optimization algorithm. The proposed two-stage scheme is tested in both direct and model predictive control settings and shows promising Pareto optimal results in terms of solution accuracy and subsequent control performance.

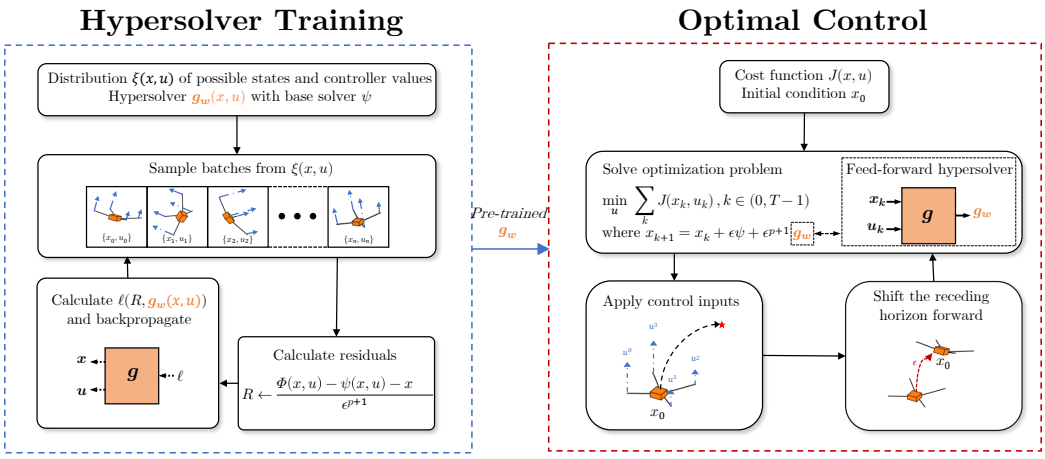

Figure 1: Overview of the proposed method. [Left] The hypersolver is trained to approximate residuals given a distribution of control inputs and states. [Right] The pre-trained hypersolver model is used to accelerate the online learning of a control policy via Model Predictive Control.

2 Numerical Optimal Control

Let us consider the following controlled dynamical system described:

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), u_\theta(t)) \\ x(0) &= x_0 \end{aligned} \tag{1}$$

with state $x \in \mathcal{X} \subset \mathbb{R}^{n_x}$, input $u_\theta \in \mathcal{U} \subset \mathbb{R}^{n_u}$ defined on a compact time domain $\mathcal{T} := [t_0, T]$ where θ is a set of free parameters of the controller. Solutions of equation 1 are denoted with $x(t) = \Phi(x(s), s, t)$ for all $s, t \in \mathcal{T}$.

2.1 Optimal Control Problem

Given some objective function $J : \mathcal{X} \times \mathcal{U} \to \mathbb{R};\ x_0, u_\theta \mapsto J(x_0, u_\theta(t))$, and a distribution $\rho_0(x_0)$ of initial conditions with support in $\mathcal{X}$, we consider the following nonlinear program, constrained to the dynamics 1:

$$\begin{aligned} \min_{u_\theta(t)} \quad & \mathbb{E}_{x_0 \sim \rho_0(x_0)} \left[J(x_0, u_\theta(t)) \right] \\ \text{subject to} \quad & \dot{x}(t) = f(t, x(t), u_\theta(t)) \\ & x(0) = x_0 \\ & t \in \mathcal{T} \end{aligned} \tag{2}$$

where the controller parameters θ are optimized. We will omit the use of θ and write $u(t) = u_\theta(t)$ from now on for simplifying the notation. Since analytic solutions of 2 exist only for limited classes of systems and objectives, numerical solvers are often applied to iteratively find a solution instead: for this reason we refer to problem 2 as *numerical optimal control*.

2.2 Solver Residuals for Controlled Dynamical Systems

Given nominal solutions Φ of equation 1 we can define the *residual* of a numerical ODE solver as the normalized error accumulated in a single step size of the method, i.e.

$$R_k = R(t_k, x(t_k), u(t_k)) = \frac{1}{\epsilon^{p+1}} \Big[\Phi(x(t_k), t_k, t_{k+1}) - x(t_k) - \epsilon \psi_\epsilon(t_k, x(t_k), u(t_k)) \Big] \tag{3}$$

where p is the order of the numerical solver corresponding to ψ_ϵ.

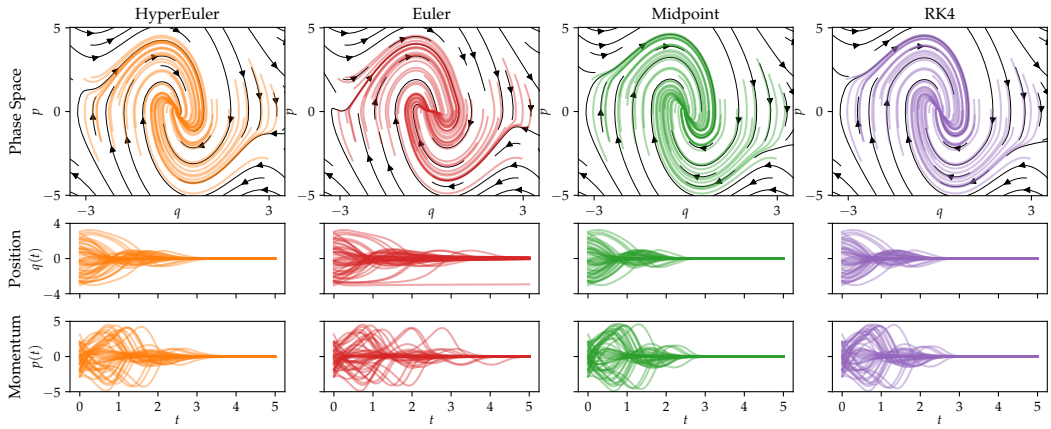

Figure 2: Pendulum trajectories simulated by an accurate adaptive solver with controllers optimized via direct optimal control. While the controller optimized with the Euler solver fails to control the system for some trajectories, the one obtained with HyperEuler can improve the performance while introducing a minimal overhead with results comparable to higher-order solvers.

3 Hypersolvers for Optimal Control

3.1 Formulation

Given a p–order base solver update map ψ_ϵ, the corresponding *hypersolver* is the discrete iteration

$$x_{k+1} = x_k + \overbrace{\epsilon\psi_\epsilon\left(t_k, x_k, u_k\right)}^{\text{base solver step}} + \epsilon^{p+1} \underbrace{g_\omega\left(t_k, x_k, u_k\right)}_{\text{approximator}} \tag{4}$$

where $g_\omega\left(t_k, x_k, u_k\right)$ is some $o(1)$ parametric function with free parameters ω. The core idea is to select g_ω as some function with *universal approximation* properties and fit the higher-order terms of the base solver by explicitly minimizing the residuals over a set of state and input samples.

3.2 Training

We assume the considered optimal control problems to feature a control input constrained in a set $\mathcal{U}$. Such constraints are often present either due to physical limitations of the actuators or safety restraints of the workspace; while this is not directly enforced nor necessary, constrained controls ensure that hypersolver pre-training is performed successfully. The analysis considers the case of *time-invariant systems* (A.3) in which dynamics of equation 1 is not directly dependent on time t, thus we rewrite with simplified notation $g_w(t, x(t), u(t)) = g_w(x, u)$. We also check the generalization properties of hypersolvers: details are available in the Appendix B.

The training procedure can be written as the following optimization problem

$$\min_{w} \quad \mathbb{E}_{(x,u)\sim\xi(x,u)} \| R(\epsilon, x, u) - g_w\left(x, u\right) \|_2 \tag{5}$$

where $\xi(x, u)$ is a distribution with support in $\mathcal{X} \times \mathcal{U}$ of the state and controller spaces. To guarantee sufficient exploration of the state-controller space, we use Monte Carlo sampling (Robert & Casella, 2013) from the given distribution.

In particular, batches of initial conditions $\{x_0^i\}$, $\{u_0^i\}$ are sampled from ξ and the optimization problem of equation 5 is solved via backpropagation for updating the parameters of the hypersolver g_w using a *stochastic gradient descent* (SGD) algorithm, e.g., `Adam` (Kingma & Ba, 2017) and we repeat the procedure for every training epoch.

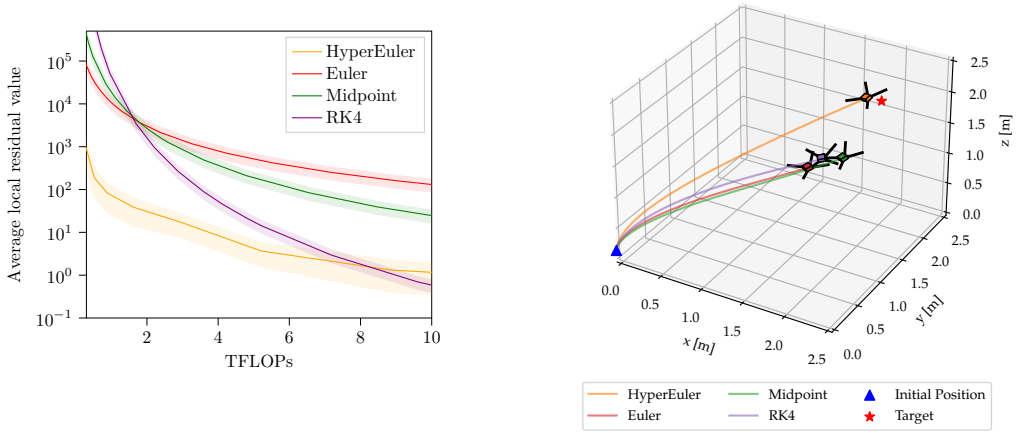

Figure 3: [Left] Pareto front of mean local residuals for the quadcopter of equation 10. The number of FLOPs (Floating Point Operations) accounts for evaluating the vector field with a neural controller and the overhead of the hypersolver. HyperEuler is Pareto efficient in FLOPS and mean local residuals for lower numbers of function evaluations. [Right] Model Predictive Control of the quadcopter with different ODE solvers. HyperEuler achieves a lower average control loss compared to the other fixed-step solvers.

3.3 Optimal Control Optimization with Hypersolvers

The pre-trained hypersolver model is employed for obtaining solutions to the trajectories of the optimal control problem in 2.1. We then optimize the control policy for minimizing the cost function J; we repeat the process iteratively until a solution is found. Figure 1 shows an overview of the proposed approach consisting in pre-training and system control.

4 Evaluation

4.1 Direct Optimal Control of a Pendulum

We consider the inverted pendulum model with a torsional spring described in equation 9. After pretraining the hypersolver by sampling from the sets of possible states $\mathcal{X}$ and controllers $\mathcal{U}$, we applied direct optimal control to optimize over full unrolled trajectories obtained via different fixed-step solvers. The *hypersolved* version of the Euler scheme, HyperEuler, improved control performance while introducing a minimal overhead as shown in Figure 3.2. Further details are available in the Appendix C.1.

4.2 Model Predictive Control of a Quadcopter

We consider the three-dimensional quadcopter model of equation 10. The controller was optimized online via the Model Predictive Control (MPC) (Garcia et al., 1989) algorithm whose receding horizon was calculated through different standard ODE solvers and our hypersolver approach. Optimization with the pre-trained HyperEuler showed a better Pareto optimality in mean local residuals and number of function evaluations compared to other fixed-step solvers while achieving the lowest cost as shown in Figure 4.2. Additional experimental details are available in the Appendix C.2.

5 Conclusion

We presented a novel method for obtaining fast and accurate control policies. Hypersolver models were firstly pre-trained on a distribution of states and controllers to approximate higher-order residuals of base fixed-step ODE solvers. The obtained models were then employed to improve the accuracy of trajectory solutions over which control policies were optimized. We verified that our method shows consistent improvements in the accuracy of ODE solutions and thus on the quality of control policies optimized through numerical solutions of the system.

References

Mato Baotić, Francesco Borrelli, Alberto Bemporad, and Manfred Morari. Efficient on-line computation of constrained optimal control. *SIAM Journal on Control and Optimization*, 47(5):2470–2489, 2008.

John Butcher. Numerical methods for differential equations and applications. *http://www.math.auckland.ac.nz/Research/Reports/view.php?id=370*, 22, 12 1997.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: Theory and practice - a survey. *Autom.*, 25:335–348, 1989.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. *Optimal control*. John Wiley & Sons, 2012.

Jacopo Panerati, Hehui Zheng, Siqi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig. Learning to fly - a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. *CoRR*, abs/2103.02142, 2021. URL https://arxiv.org/abs/2103.02142.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models, 2020a.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Torchdyn: A neural differential equations library, 2020b.

Christian Robert and George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

I Michael Ross and Fariba Fahroo. Issues in the real-time computation of optimal control. *Mathematical and computer modelling*, 43(9-10):1172–1188, 2006.

Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions, 2020.

Guillem Torrente, Elia Kaufmann, Philipp Foehn, and Davide Scaramuzza. Data-driven MPC for quadrotors. *CoRR*, abs/2102.05773, 2021. URL https://arxiv.org/abs/2102.05773.

S Vadali, Hanspeter Schaub, and K Alfriend. Initial conditions and fuel-optimal control for formation flying of satellites. In *Guidance, Navigation, and Control Conference and Exhibit*, pp. 4265, 1999.

Gerhard Wanner and Ernst Hairer. *Solving ordinary differential equations II*, volume 375. Springer Berlin Heidelberg, 1996.

Yue J Zhang, Andreas A Malikopoulos, and Christos G Cassandras. Optimal control and coordination of connected and automated vehicles at urban traffic intersections. In *2016 American Control Conference (ACC)*, pp. 6227–6232. IEEE, 2016.

Liu Ziyin, Tilman Hartwig, and Masahito Ueda. Neural networks fail to learn periodic functions and how to fix it, 2020.

A Additional Hypersolver Material

A.1 Bounds on Solution Errors

From the definition of residual in equation 3, we can define the *local truncation error* $e_k := \left\|\epsilon^{p+1} R_k\right\|_2$. Given a p-th order explicit solver, we have $e_k = o(\epsilon^{p+1})$ (Butcher, 1997). For hypersolvers, similar error bounds are defined as $e_k \leq o(\delta\epsilon^{p+1})$, where $\delta > 0$ depends on the hypersolver training results (Poli et al., 2020a, Theorem 1). This result practically guarantees that if g_ω is a good approximator for R, i.e. $\delta \ll 1$, then the overall local truncation error of the *hypersolved* ODE is significantly reduced.

A.2 Explicit HyperEuler Formulation

During the experiments, we restricted our analysis to the *hypersolved* version of the Euler scheme, namely `HyperEuler` because of the highest potential speed-up it can achieve: since Euler is a first-order method, it requires the least number of function evaluations (NFE) of the vector field f in equation 1 and yields a second order local truncation error $e_k := \left\|\epsilon^2 R_k\right\|_2$, which is larger than other fixed-step solvers. In particular, the base solver scheme ψ of equation 4 can be written as $\psi_\epsilon\left(t_k, x_k, u_k\right) = f\left(t_k, x_k, u_k\right)$, which is approximating the next state by adding an evaluation of the vector field multiplied by the step size ϵ. The `HyperEuler` scheme can be explicitly written as

$$\begin{cases} x_{k+1} = x_k + \epsilon f(t_k, x_k, u_k) + \epsilon^2 g_w\left(t_k, x_k, u_k\right) \\ x_{k=0} = x_0 \end{cases} \tag{6}$$

while its residual can be written as

$$R\left(x(t_k), u(t_k)\right)) = \Phi(x(t_k), t_k, t_{k+1}) - x(t_k) - \epsilon f(t_k, x_k, u_k) \tag{7}$$

A.3 Hypersolvers for Time-invariant Systems

A *time-invariant* system with time-invariant controller can be described as following

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(x(t))) \\ x(0) &= x_0 \end{aligned} \tag{8}$$

in which f and u do not explicitly depend on time.

A.4 Architecture

We designed the hypersolver network g_w as a feed–forward neural network. Table 1 summarizes the parameters used for the considered controlled systems, where activation functions `SoftPlus`: $x \mapsto \log(1 + e^x)$ and `Tanh`: $x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

Table 1: Parameters for hypersolver network

	Inverted Pendulum	Quadcopter
Input Layer	5	46
Hidden Layer 1	32	64
Activation Function 1	Softplus	Softplus
Hidden Layer 2	32	64
Activation Function 2	Tanh	Softplus
Output Layer	2	21

We also used the vector field f as an input of the hypersolver, which, however, does not require a further evaluation since it is pre-evaluated at runtime by the base solver ψ. We emphasize that the size of the network should depend on the application: a too-large neural network may require more computations than just increasing the numerical solver's complexity: Pareto optimality of hypersolvers also depends on their complexity. By keeping their neural network small enough, we can guarantee that evaluating the hypersolvers is cheaper than resorting to more complex numerical routines.

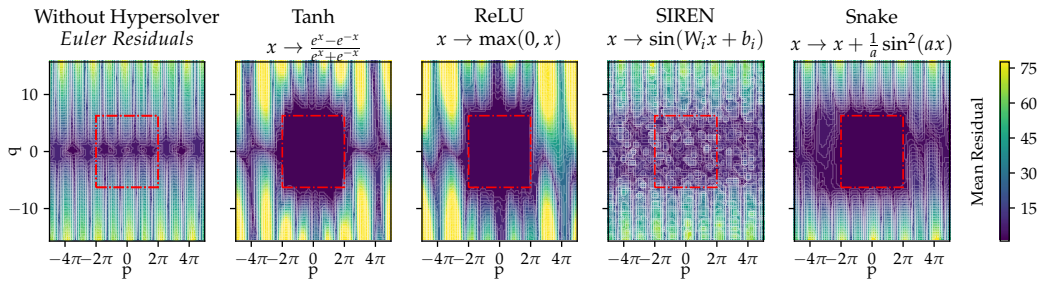

Figure 4: Generalization outside of the training region (red rectangle) of the state space with different hypersolver activation functions. Architectures containing activation functions with periodic components achieve better extrapolation properties compared to the others.

B Generalization Study

We assumed the state and controller spaces to be bounded and that training be performed sampling for their known distributions. While this is sufficient for optimal control problems given *a priori* known bounds, we also investigate how the system generalizes to unseen states and control input values. In particular, we found that activation functions have an impact on the end result of generalization beyond known boundaries. We took into consideration two commonly used activation functions, $\texttt{Tanh} : x \to \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and $\texttt{ReLU} : x \to \max(0, x)$, along with network architectures which employ activation functions containing periodic components: $\texttt{SIREN} : x \to \sin Wx + b$ (Sitzmann et al., 2020) and $\texttt{Snake} : x \to x + \frac{1}{a}\sin ax^2$ (Ziyin et al., 2020). We trained hypersolver models with the different activation functions for the inverted pendulum of equation 9 with common settings: time step ϵ was chosen as $0.1\ [s]$, networks were trained for 100000 epochs with the `Adam` optimizer and learning rate of 10^{-3}, where sampling was done on from the uniform distribution $\xi(x, u)$ with support in $\mathcal{X} \times \mathcal{U}$ with $\mathcal{X} = [-2\pi, 2\pi] \times [-2\pi, 2\pi]$ and $\mathcal{U} = [-10, 10]$. Figure 4 and Figure 5 show generalization outside the states and controllers distribution respectively. We notice that while `Tanh` and `ReLU` perform well on the training set of interest, performance degrades rapidly outside of it. On the other one hand, `SIREN` and `Snake` manage to extrapolate the periodicity of the residual distribution even outside of the training region, thus providing further empirical evidence of the universal approximation theorem (Ziyin et al., 2020, Theorem 3).

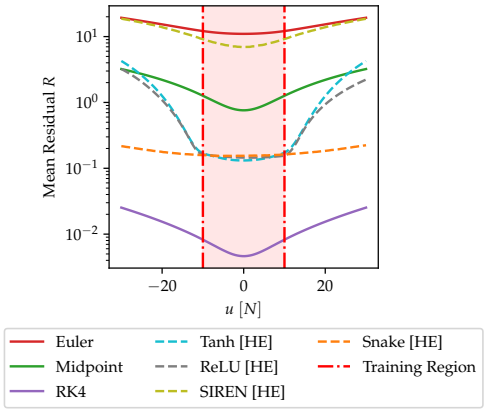

Figure 5: Generalization outside of the training region (red lines) in the controller space with different hypersolver activation functions. The `Snake` activation function manages to extrapolate best to unseen controllers compared to others.

C Experimental Details

C.1 Inverted Pendulum

System Dynamics We model the inverted pendulum with elastic joint with Hamiltonian dynamics via the following:

$$\begin{bmatrix} \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} 0 & 1/m \\ -k & -\beta/m \end{bmatrix} \begin{bmatrix} q \\ p \end{bmatrix} - \begin{bmatrix} 0 \\ mgl \sin q \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{9}$$

where $m = 1\ [Kg]$, $k = 0.5\ [N/\text{rad}]$, $r = 1\ [m]$, $\beta = 0.01\ [Ns/\text{rad}]$, $g = 9.81\ [m/s^2]$.

Pretraining We selected $\xi(x, u)$ as a uniform distribution with support in $\mathcal{X} \times \mathcal{U}$ where $\mathcal{X} = [-2\pi, 2\pi] \times [-2\pi, 2\pi]$ and $\mathcal{U} = [-5, 5]$ to guarantee sufficient exploration of the state-controller space. We trained the hypersolver on local residuals using the `Adam` optimizer with learning rate of 3×10^{-4} for $300,000$ epochs.

Direct Optimal Control The goal was to stabilize the inverted pendulum of equation 9 in the vertical position. We chose $t \in [0, 3]$ and a step size $\epsilon = 0.2\ s$ for the experiment. The control input was assumed continuously time-varying. Nominal trajectories by the controller obtained with Euler showed around a 50% degradation in performance due to the higher cost function value compared to RK4, while the ones with HyperEuler only had a 7% degradation in performance while only adding less than 2% of computational overhead compared to the Euler baseline in terms of FLOPs.

C.2 Quadcopter

System Dynamics The quadcopter model is a suitably modified version of the explicit dynamics update in (Panerati et al., 2021) for batched training in PyTorch. We do not consider for the sake of the experiment the downwash (effect of other drones) and ground effect; the latter is analyzed in (Torrente et al., 2021) in which a nominal dynamics model is combined with a neural network modeling the ground effect to obtain a robust data-driven MPC. The dynamic model is described by the following accelerations update:

$$\begin{aligned} \ddot{\mathbf{x}} &= \left(\mathbf{R} \cdot [0, 0, k_F \textstyle\sum_{i=0}^{3} \omega_i^2] - [0, 0, mg]\right) m^{-1} \\ \ddot{\boldsymbol{\psi}} &= \mathbf{J}^{-1} \left(\tau(l, k_F, k_T, [\omega_0^2, \omega_1^2, \omega_2^2, \omega_3^2]) - \dot{\boldsymbol{\psi}} \times \left(\mathbf{J}\dot{\boldsymbol{\psi}}\right)\right) \end{aligned} \tag{10}$$

where $\mathbf{x} = [x, y, z]$ corresponds to the drone positions and $\boldsymbol{\psi} = [\phi, \theta, \psi]$ to its angular positions; $\boldsymbol{R}$ and $\boldsymbol{J}$ are its rotation and inertial matrices respectively, $\tau(\cdot)$ is a function calculating the torques induced by the motor speeds ω_i, while arm length l, mass m, gravity acceleration constant g along with k_F and k_T are scalar variables describing the quadcopter's physical properties.

Pretraining We chose $\xi(x, u)$ as a uniform distribution with support in $\mathcal{X} \times \mathcal{U}$ where $\mathcal{X}$ was selected to cover the possible values of states at runtime. Each of the four motors $i \in [0, 3]$ has control inputs $u^i \in [75, 925]$ `rpm` to guarantee sufficient exploration of the state-controller space. We trained the hypersolver on local residuals using the `Adam` optimizer with learning rate of 3×10^{-4} for $300,000$ epochs.

Model Predictive Control The task was to move the quadcopter of equation 10 from the origin to target position $\mathbf{x}^\star = [2, 2, 2]$, while we also penalized large values for angular positions and velocities. We chose $t \in [0, 2]$ and a step size $\epsilon = 0.02\ s$. The control input was assumed piece-wise constant between sampling times. The hypersolver model was verified to be Pareto optimal with respect to FLOPS and mean residuals for lower number of function evaluations. The model predictive control experiment showed that the controller whose receding horizon was calculated through the HyperEuler scheme achieved a mean lower value of the cost function of 4% than the 4-th order RK4 which resulted in HyperEuler approaching the target position faster compared to the other solvers.

C.3 Hardware and Software

Experiments were carried out on a machine equipped with an AMD RYZEN THREADRIPPER 3960X CPU and two NVIDIA RTX 3090 graphic cards. Software-wise, we used `PyTorch` (Paszke et al., 2019) for deep learning and the `torchdyn` (Poli et al., 2020b) and `torchdiffeq` (Chen et al., 2019) libraries for ODE solvers.