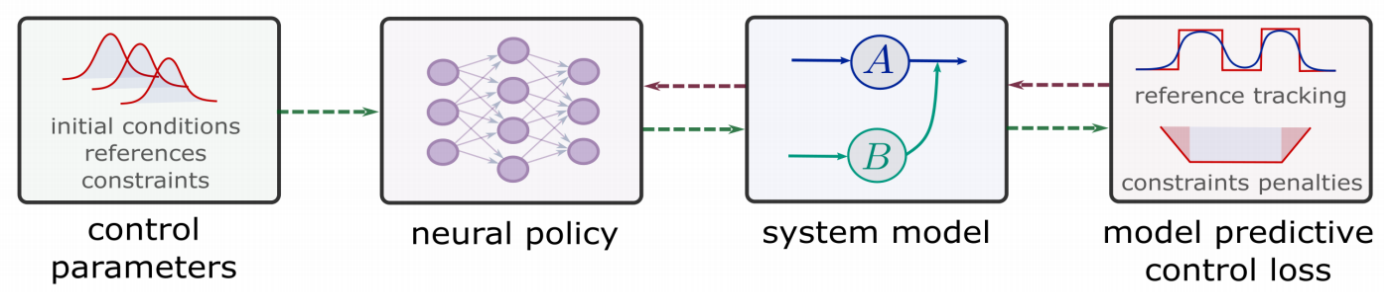


SciML Tutorial: Learning to Control (L2C)

Ján Drgoňa
Associate Professor
Civil and Systems Engineering Department
Electrical and Computer Engineering Department (secondary)
The Ralph O'Connor Sustainable Energy Institute (ROSEI)
Data Science and AI Institute (DSAI)
Johns Hopkins University

8th Annual Learning for Dynamics & Control Conference
University of Southern California, June 17

Learning to Control (L2C) Problem Formulation

Policy Learning for Parametric Optimal Control

Consider the finite horizon parametric optimal control problem

$$\begin{aligned} \min_{\{u_{0:N-1}\}} \quad & J(x_0, \xi) = \sum_{k=0}^{N-1} \ell(x_k, u_k; \xi) + \ell_N(x_N; \xi) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k; \xi), \\ & g(x_k, u_k; \xi) \leq 0, \\ & h(x_k, u_k; \xi) = 0. \end{aligned}$$

The optimal control solution operator is $\Pi^\star : (x_k, \xi) \mapsto u_k^\star(x_k; \xi)$.
L2C aims to learn a parameterized policy $\pi_\theta : (x_k, \xi) \mapsto u_k$, that approximates $\Pi^\star$ across operating conditions $x_0 \sim \mathcal{X}_0$ and $\xi \sim \mathcal{D}$.

Learning to Control (L2C) Problem Formulation

Policy Learning for Parametric Optimal Control

Consider the finite horizon parametric optimal control problem

$$\min_{\{u_{0:N-1}\}} \quad J(x_0, \xi) = \sum_{k=0}^{N-1} \ell(x_k, u_k; \xi) + \ell_N(x_N; \xi)$$

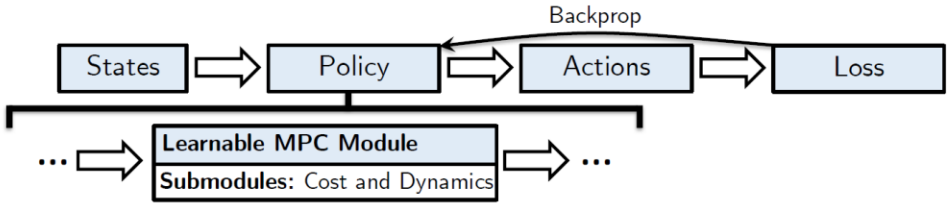
$$\text{s.t.} \quad x_{k+1} = f(x_k, u_k; \xi),$$
$$g(x_k, u_k; \xi) \leq 0,$$
$$h(x_k, u_k; \xi) = 0.$$

Assume we have differentiable cost function terms, system dynamics, constraints, and known distributions of scenarios.

The optimal control solution operator is $\Pi^\star : (x_k, \xi) \mapsto u_k^\star(x_k; \xi)$.
L2C aims to learn a parameterized policy $\pi_\theta : (x_k, \xi) \mapsto u_k$, that approximates $\Pi^\star$ across operating conditions $x_0 \sim \mathcal{X}_0$ and $\xi \sim \mathcal{D}$.

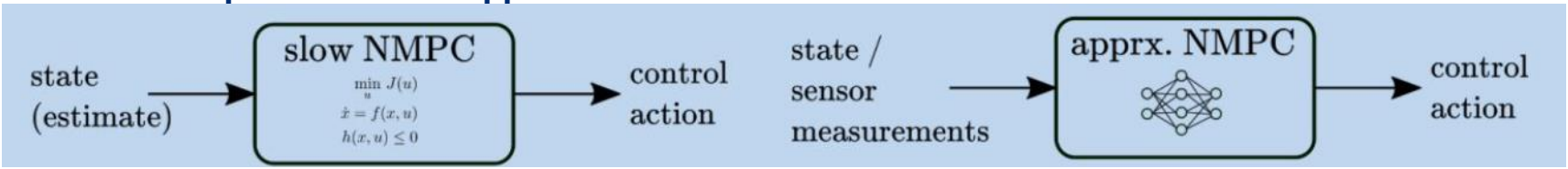
Model-Based Learning to Control (L2C) Methodologies

Supervised L2C: Differentiable Model Predictive Control



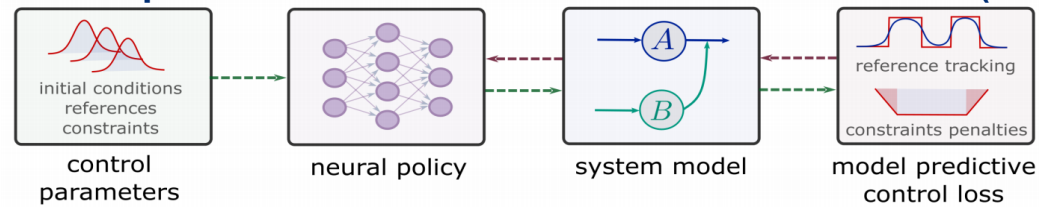
Brandon Amos, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter, Differentiable MPC for End-to-end Planning and Control, NeurIPS 2018.
Jonathan Frey, et al., Differentiable Nonlinear Model Predictive Control, arXiv:2505.01353, 2025

Supervised L2C: Approximate Model Predictive Control



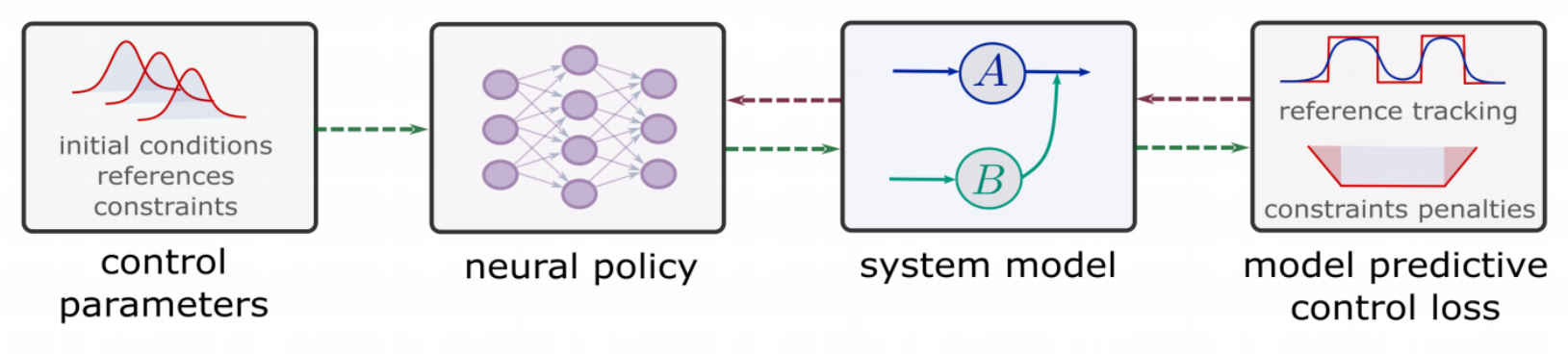
M. Hertneck, et al., Learning an Approximate Model Predictive Controller With Guarantees, in IEEE Control Systems Letters, 2018
B. Karg and S. Lucia, Efficient Representation and Approximation of Model Predictive Control Laws via Deep Learning, in IEEE Transactions on Cybernetics, 2020
S Chen, et al. Approximating explicit model predictive control using constrained neural networks, ACC, 2018

Self-Supervised L2C: Differentiable Predictive Control (DPC)

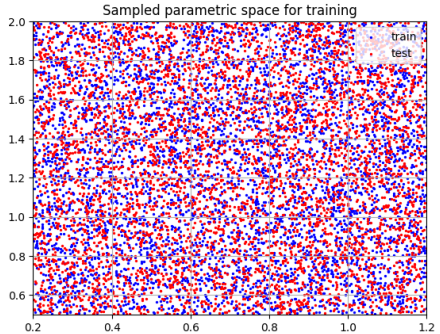


J. Drgoňa, A. Tuor and D. Vrabie, Learning Constrained Parametric Differentiable Predictive Control Policies With Guarantees, in IEEE TSMC, 2024

Self-Supervised L2C: Differentiable Predictive Control (DPC)



Dataset: collocation points in the control parametric space.



Architecture: differentiable model with neural network control policy.

$$x_{k+1} = \text{ODESolve}(f(x_k, u_k))$$
$$u_k = NN_\theta(x_k, \xi_k)$$
$$g(x_k, u_k, \xi_k) \leq 0$$
$$x_0 \sim \mathcal{P}_{x_0}$$
$$\xi_k \sim \mathcal{P}_\xi$$

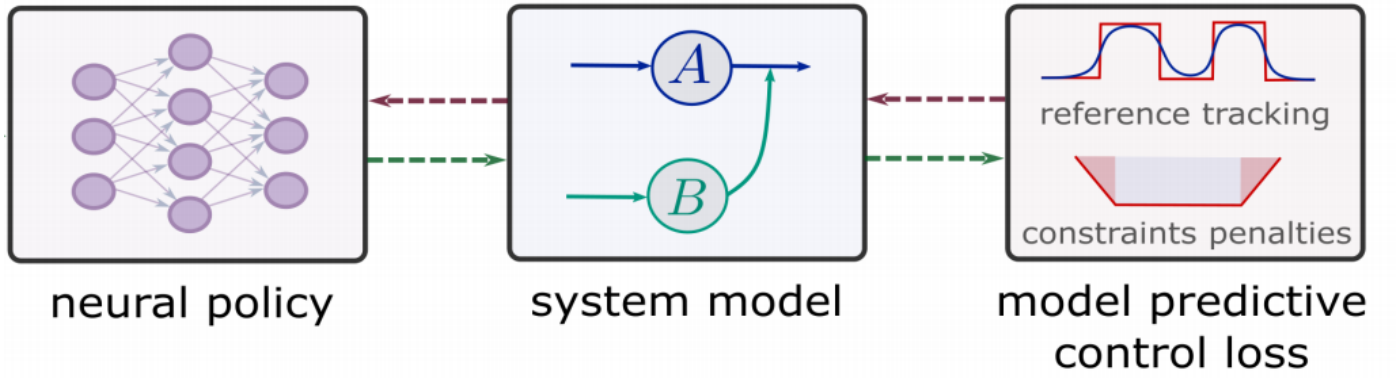
Loss function: reference tracking, constraints and terminal penalties.

$$\ell_1 = \sum_{i=1}^{m} \sum_{k=1}^{N-1} Q_x ||x_k^i - r_k^i||_2^2$$
$$\ell_2 = \sum_{i=1}^{m} \sum_{k=1}^{N-1} Q_g ||\texttt{RELU}(g(x_k^i, u_k^i, \xi_k^i)||_2^2$$
$$\ell_{L2C} = \ell_1 + \ell_2$$

https://github.com/pnnl/neuromancer/blob/master/examples/control/Part_3_ref_tracking_ODE.ipynb

J. Drgoňa, A. Tuor and D. Vrabie, "Learning Constrained Parametric Differentiable Predictive Control Policies With Guarantees," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2024

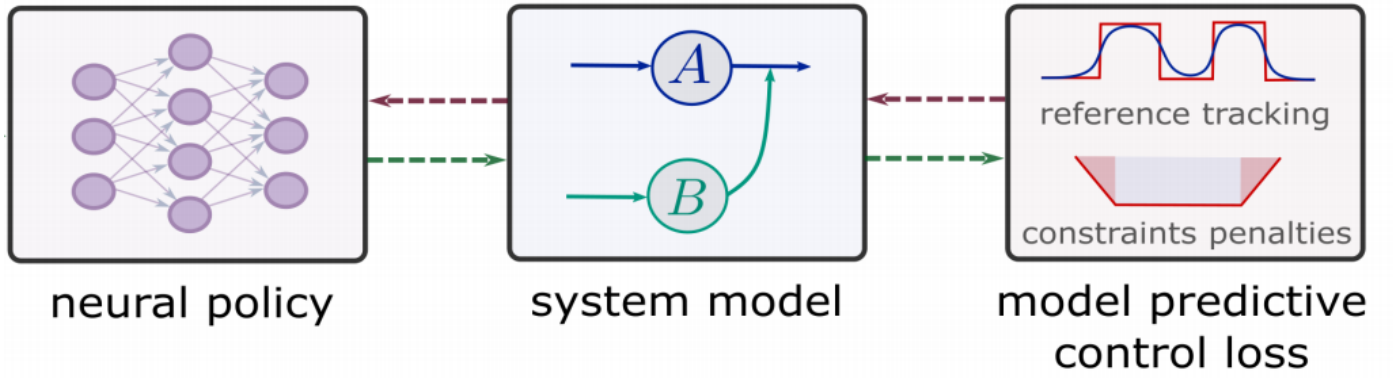
Differentiable Closed-Loop System



Forward pass. For a single scenario, the closed-loop recursion and loss are

$$
\begin{aligned}
u_k &= \pi_\theta(x_k;\xi_k),\\
x_{k+1} &= f(x_k,u_k;\xi_k),\\
\ell_k &= \ell(x_k,u_k;\xi_k) \;+\; p_x\big(g(x_k;\xi_k)\big) \;+\; p_u\big(h(u_k;\xi_k)\big),\\
\mathcal{L} &= \sum_{k=0}^{N-1} \ell_k \;+\; \ell_N(x_N;\xi_N).
\end{aligned}
$$

Differentiable Closed-Loop System



Forward pass. For a single scenario, the closed-loop recursion and loss are

$$
\begin{aligned}
u_k &= \pi_\theta(x_k;\xi_k),\\
x_{k+1} &= f(x_k,u_k;\xi_k),\\
\ell_k &= \ell(x_k,u_k;\xi_k) \;+\; p_x\big(g(x_k;\xi_k)\big) \;+\; p_u\big(h(u_k;\xi_k)\big),\\
\mathcal{L} &= \sum_{k=0}^{N-1} \ell_k \;+\; \ell_N(x_N;\xi_N).
\end{aligned}
$$

From an RL perspective, the DPC loss can be seen as a white-box value function suitable for gradient-based policy optimization.

Differentiable Closed-Loop System

Backward pass (sensitivities). Let

$$A_k := \frac{\partial f}{\partial x}, \quad B_k := \frac{\partial f}{\partial u}, \quad P_k := \frac{\partial \pi_\theta}{\partial x}, \quad G_k := \frac{\partial \pi_\theta}{\partial \theta} \quad \text{(all evaluated at } (x_k, u_k; \xi_k)).$$

Define $s_N := \frac{\partial \ell_N}{\partial x}(x_N; \xi_N)$ and stage gradients

$$\ell_{x,k} := \frac{\partial \ell}{\partial x} + \frac{\partial p_x}{\partial x}, \qquad \ell_{u,k} := \frac{\partial \ell}{\partial u} + \frac{\partial p_u}{\partial u}.$$

Then the reverse recursion for $k = N-1, \ldots, 0$ is

$$s_k \;=\; \ell_{x,k} \;+\; A_k^\top s_{k+1} \;+\; P_k^\top \big(\ell_{u,k} + B_k^\top s_{k+1}\big),$$

and the gradient w.r.t. the policy parameters is

$$\nabla_\theta \mathcal{L} \;=\; \sum_{k=0}^{N-1} G_k^\top \big(\ell_{u,k} + B_k^\top s_{k+1}\big).$$

Differentiable Closed-Loop System

Backward pass (sensitivities). Let

$$A_k := \frac{\partial f}{\partial x}, \quad B_k := \frac{\partial f}{\partial u}, \quad P_k := \frac{\partial \pi_\theta}{\partial x}, \quad G_k := \frac{\partial \pi_\theta}{\partial \theta} \quad \text{(all evaluated at } (x_k, u_k; \xi_k)).$$

Define $s_N := \frac{\partial \ell_N}{\partial x}(x_N; \xi_N)$ and stage gradients

$$\ell_{x,k} := \frac{\partial \ell}{\partial x} + \frac{\partial p_x}{\partial x}, \qquad \ell_{u,k} := \frac{\partial \ell}{\partial u} + \frac{\partial p_u}{\partial u}.$$

Then the reverse recursion for $k = N-1, \ldots, 0$ is

$$s_k \;=\; \ell_{x,k} \;+\; A_k^\top s_{k+1} \;+\; P_k^\top \big(\ell_{u,k} + B_k^\top s_{k+1}\big),$$

and the gradient w.r.t. the policy parameters is

$$\nabla_\theta \mathcal{L} \;=\; \sum_{k=0}^{N-1} G_k^\top \big(\ell_{u,k} + B_k^\top s_{k+1}\big).$$

The backward pass in DPC is mathematically identical to **Backpropagation Through Time (BPTT)** used to train recurrent neural networks.

Differentiable Closed-Loop System

Backward pass (sensitivities). Let

$$A_k := \frac{\partial f}{\partial x}, \quad B_k := \frac{\partial f}{\partial u}, \quad P_k := \frac{\partial \pi_\theta}{\partial x}, \quad G_k := \frac{\partial \pi_\theta}{\partial \theta} \quad \text{(all evaluated at } (x_k, u_k; \xi_k)).$$

Define $s_N := \frac{\partial \ell_N}{\partial x}(x_N; \xi_N)$ and stage gradients

$$\ell_{x,k} := \frac{\partial \ell}{\partial x} + \frac{\partial p_x}{\partial x}, \qquad \ell_{u,k} := \frac{\partial \ell}{\partial u} + \frac{\partial p_u}{\partial u}.$$

Then the reverse recursion for $k = N-1, \ldots, 0$ is

$$s_k \;=\; \ell_{x,k} \;+\; A_k^\top s_{k+1} \;+\; P_k^\top \big(\ell_{u,k} + B_k^\top s_{k+1}\big),$$

and the gradient w.r.t. the policy parameters is

$$\nabla_\theta \mathcal{L} \;=\; \sum_{k=0}^{N-1} G_k^\top \big(\ell_{u,k} + B_k^\top s_{k+1}\big).$$

The backward pass in DPC computes discrete adjoint sensitivities **leading to Pontryagin adjoint equations** in continuous time limit**.**

DPC Policy Optimization Algorithm

Algorithm 5: Self-Supervised DPC Policy Optimization

Input: Horizon N, batch size B; initial-state distribution $\mathcal{X}_0$, scenario distribution $\mathcal{D}$; dynamics $x_{k+1} = f(x_k, u_k; \xi_k)$; policy class $u_k = \pi_\theta(x_k; \xi_k)$; costs ℓ, ℓ_N; penalties p_x, p_u with weights Q_x, Q_u; constraints g, h; optimizer $\mathbb{O}$

Output: Trained explicit policy $\pi_{\theta^\star}$

while *not converged* **do**

  Sample mini-batch $\{(x_0^{(i)}, \xi^{(i)})\}_{i=1}^{B} \sim \mathcal{X}_0 \times \mathcal{D}$;

  for $i \leftarrow 1$ **to** B **do**

    for $k \leftarrow 0$ **to** $N-1$ **do**

      $u_k^{(i)} \leftarrow \pi_\theta\big(x_k^{(i)}; \xi_k^{(i)}\big)$;

      $x_{k+1}^{(i)} \leftarrow f\big(x_k^{(i)}, u_k^{(i)}; \xi_k^{(i)}\big)$;

      $\ell_k^{(i)} \leftarrow \ell\big(x_k^{(i)}, u_k^{(i)}; \xi_k^{(i)}\big) + Q_x\, p_x\big(g(x_k^{(i)}; \xi_k^{(i)})\big) + Q_u\, p_u\big(h(u_k^{(i)}; \xi_k^{(i)})\big)$;

    $\mathcal{L}^{(i)} \leftarrow \frac{1}{N} \sum_{k=0}^{N-1} \ell_k^{(i)} + \ell_N\big(x_N^{(i)}; \xi_N^{(i)}\big)$;

  $\mathcal{L} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}^{(i)}$; // batch DPC loss

  Compute $\nabla_\theta \mathcal{L}$ e.g., BPTT or adjoint sensitivity

  $\theta \leftarrow \mathbb{O}\big(\theta, \nabla_\theta \mathcal{L}\big)$; // e.g., Adam step

return π_θ;

DPC Policy Optimization Algorithm

Algorithm 5: Self-Supervised DPC Policy Optimization

Input: Horizon N, batch size B; initial-state distribution $\mathcal{X}_0$, scenario distribution $\mathcal{D}$; dynamics $x_{k+1} = f(x_k, u_k; \xi_k)$; policy class $u_k = \pi_\theta(x_k; \xi_k)$; costs ℓ, ℓ_N; penalties p_x, p_u with weights Q_x, Q_u; constraints g, h; optimizer $\mathbb{O}$

Output: Trained explicit policy $\pi_{\theta^\star}$

while *not converged* **do**

  Sample mini-batch $\{(x_0^{(i)}, \xi^{(i)})\}_{i=1}^{B} \sim \mathcal{X}_0 \times \mathcal{D}$;

  for $i \leftarrow 1$ **to** B **do**

    for $k \leftarrow 0$ **to** $N-1$ **do**

      $u_k^{(i)} \leftarrow \pi_\theta\big(x_k^{(i)}; \xi_k^{(i)}\big)$;

      $x_{k+1}^{(i)} \leftarrow f\big(x_k^{(i)}, u_k^{(i)}; \xi_k^{(i)}\big)$;

      $\ell_k^{(i)} \leftarrow \ell\big(x_k^{(i)}, u_k^{(i)}; \xi_k^{(i)}\big) + Q_x\, p_x\big(g(x_k^{(i)}; \xi_k^{(i)})\big) + Q_u\, p_u\big(h(u_k^{(i)}; \xi_k^{(i)})\big)$;

    $\mathcal{L}^{(i)} \leftarrow \frac{1}{N} \sum_{k=0}^{N-1} \ell_k^{(i)} + \ell_N\big(x_N^{(i)}; \xi_N^{(i)}\big)$;

  $\mathcal{L} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}^{(i)}$; // batch DPC loss

  Compute $\nabla_\theta \mathcal{L}$ e.g., BPTT or adjoint sensitivity

  $\theta \leftarrow \mathbb{O}\big(\theta, \nabla_\theta \mathcal{L}\big)$; // e.g., Adam step

return π_θ;

Forward pass

DPC Policy Optimization Algorithm

Algorithm 5: Self-Supervised DPC Policy Optimization

Input: Horizon N, batch size B; initial-state distribution $\mathcal{X}_0$, scenario distribution $\mathcal{D}$; dynamics $x_{k+1} = f(x_k, u_k; \xi_k)$; policy class $u_k = \pi_\theta(x_k; \xi_k)$; costs ℓ, ℓ_N; penalties p_x, p_u with weights Q_x, Q_u; constraints g, h; optimizer $\mathbb{O}$

Output: Trained explicit policy $\pi_{\theta^\star}$

while *not converged* **do**

  Sample mini-batch $\{(x_0^{(i)}, \xi^{(i)})\}_{i=1}^{B} \sim \mathcal{X}_0 \times \mathcal{D}$;

  for $i \leftarrow 1$ **to** B **do**

    for $k \leftarrow 0$ **to** $N-1$ **do**

      $u_k^{(i)} \leftarrow \pi_\theta\big(x_k^{(i)}; \xi_k^{(i)}\big)$;

      $x_{k+1}^{(i)} \leftarrow f\big(x_k^{(i)}, u_k^{(i)}; \xi_k^{(i)}\big)$;

      $\ell_k^{(i)} \leftarrow \ell\big(x_k^{(i)}, u_k^{(i)}; \xi_k^{(i)}\big) + Q_x\, p_x\big(g(x_k^{(i)}; \xi_k^{(i)})\big) + Q_u\, p_u\big(h(u_k^{(i)}; \xi_k^{(i)})\big)$;

    $\mathcal{L}^{(i)} \leftarrow \frac{1}{N} \sum_{k=0}^{N-1} \ell_k^{(i)} + \ell_N\big(x_N^{(i)}; \xi_N^{(i)}\big)$;

  $\mathcal{L} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}^{(i)}$; // batch DPC loss

  Compute $\nabla_\theta \mathcal{L}$ e.g., BPTT or adjoint sensitivity

  $\theta \leftarrow \mathbb{O}\big(\theta, \nabla_\theta \mathcal{L}\big)$; // e.g., Adam step

return π_θ;

Backward pass
Gradient step

DPC Policy Optimization Algorithm

Algorithm 5: Self-Supervised DPC Policy Optimization

Input: Horizon N, batch size B; initial-state distribution $\mathcal{X}_0$, scenario distribution $\mathcal{D}$; dynamics $x_{k+1} = f(x_k, u_k; \xi_k)$; policy class $u_k = \pi_\theta(x_k; \xi_k)$; costs ℓ, ℓ_N; penalties p_x, p_u with weights Q_x, Q_u; constraints g, h; optimizer $\mathbb{O}$

Output: Trained explicit policy $\pi_{\theta^\star}$

while *not converged* **do**

  Sample mini-batch $\{(x_0^{(i)}, \xi^{(i)})\}_{i=1}^{B} \sim \mathcal{X}_0 \times \mathcal{D}$;

  for $i \leftarrow 1$ **to** B **do**

    for $k \leftarrow 0$ **to** $N-1$ **do**

      $u_k^{(i)} \leftarrow \pi_\theta\big(x_k^{(i)}; \xi_k^{(i)}\big)$;

      $x_{k+1}^{(i)} \leftarrow f\big(x_k^{(i)}, u_k^{(i)}; \xi_k^{(i)}\big)$;

      $\ell_k^{(i)} \leftarrow \ell\big(x_k^{(i)}, u_k^{(i)}; \xi_k^{(i)}\big) + Q_x\, p_x\big(g(x_k^{(i)}; \xi_k^{(i)})\big) + Q_u\, p_u\big(h(u_k^{(i)}; \xi_k^{(i)})\big)$;

    $\mathcal{L}^{(i)} \leftarrow \frac{1}{N} \sum_{k=0}^{N-1} \ell_k^{(i)} + \ell_N\big(x_N^{(i)}; \xi_N^{(i)}\big)$;

  $\mathcal{L} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}^{(i)}$; // batch DPC loss

  Compute $\nabla_\theta \mathcal{L}$ e.g., BPTT or adjoint sensitivity

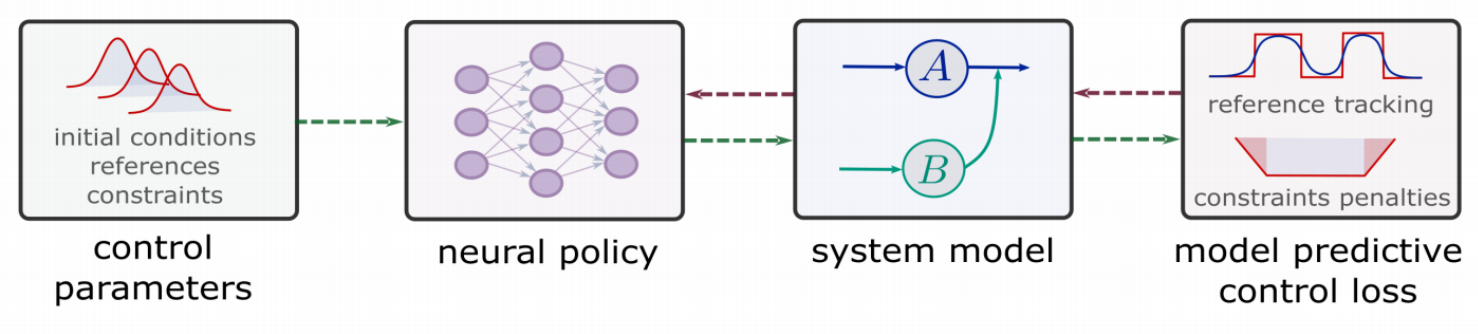
  $\theta \leftarrow \mathbb{O}\big(\theta, \nabla_\theta \mathcal{L}\big)$; // e.g., Adam step

return π_θ;

DPC learns approximate explicit MPC policies through sampling-based self-supervised policy optimization.

No MPC demonstrations required.
No learned value function.
No expensive online optimization.
Offline computation is amortized across a distribution of problems.

DPC Example in NeuroMANCER SciML Library



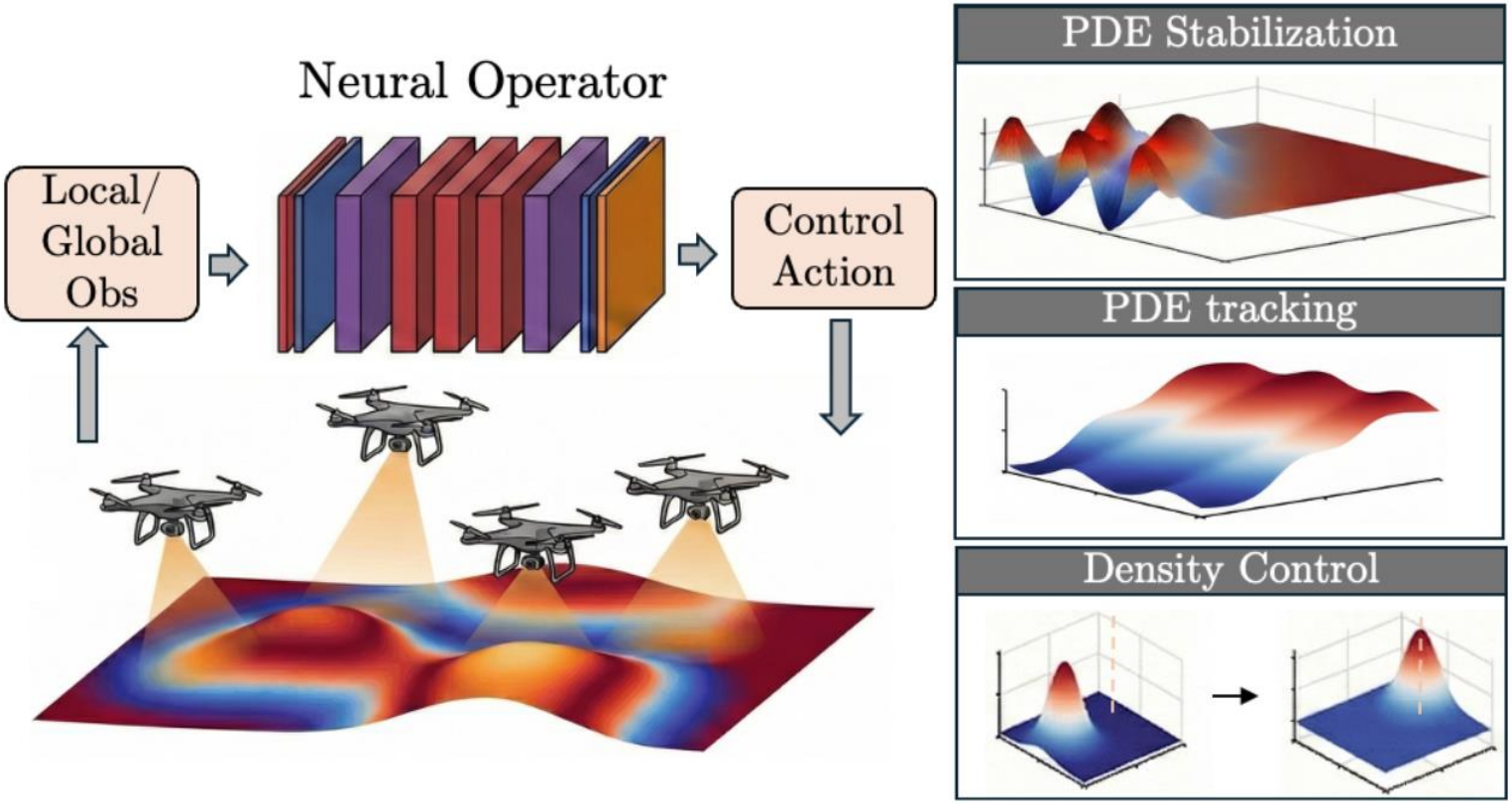
colab.research.google.com/github/SciML-L4DC2026/SciML-L4DC2026/blob/main/notebooks/L2C/Example_1_DPC_ODE/Example_1_DPC_ODE.ipynb

sciml-l4dc2026.github.io/SciML-L4DC2026

github.com/pnnl/neuromancer

NEUROMANCER
Scientific Machine Learning

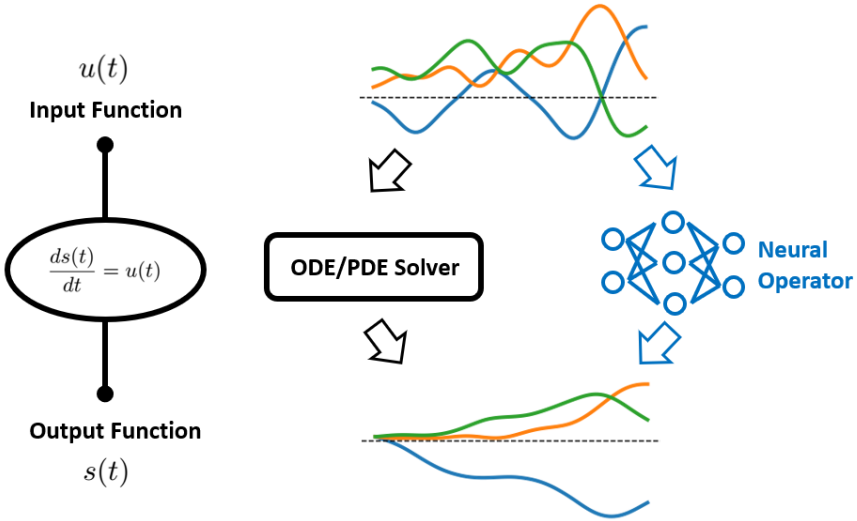
Learning to Control Partial Differential Equations with Neural Operators



P. Zanotta, H. Zheng, D. R. Sharkar, S. Goswami, J. Drgoňa, Cardinality-Invariant Neural Operator Policies for Scalable PDE Control, ICML, 2026, https://solaris-jhu.github.io/CINOC-website/

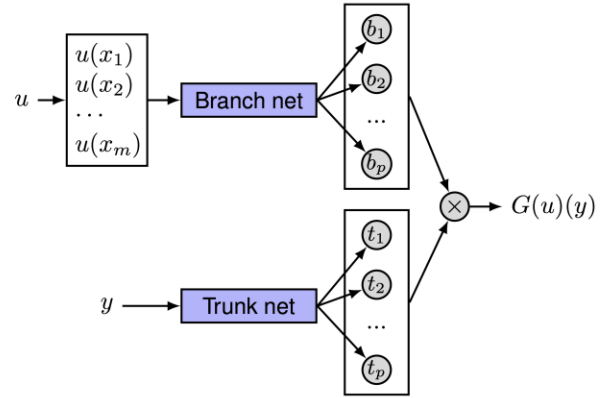
Neural Operators in a Nutshell

Neural networks learn maps between vector spaces.
In contrast, neural operators learn maps between infinite-dimensional function spaces.

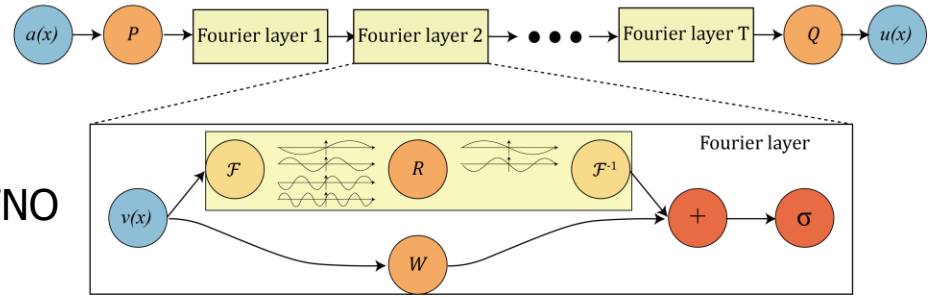


https://towardsdatascience.com/operator-learning-via-physics-informed-deeponet-lets-implement-it-from-scratch-6659f3179887/

DeepOnet

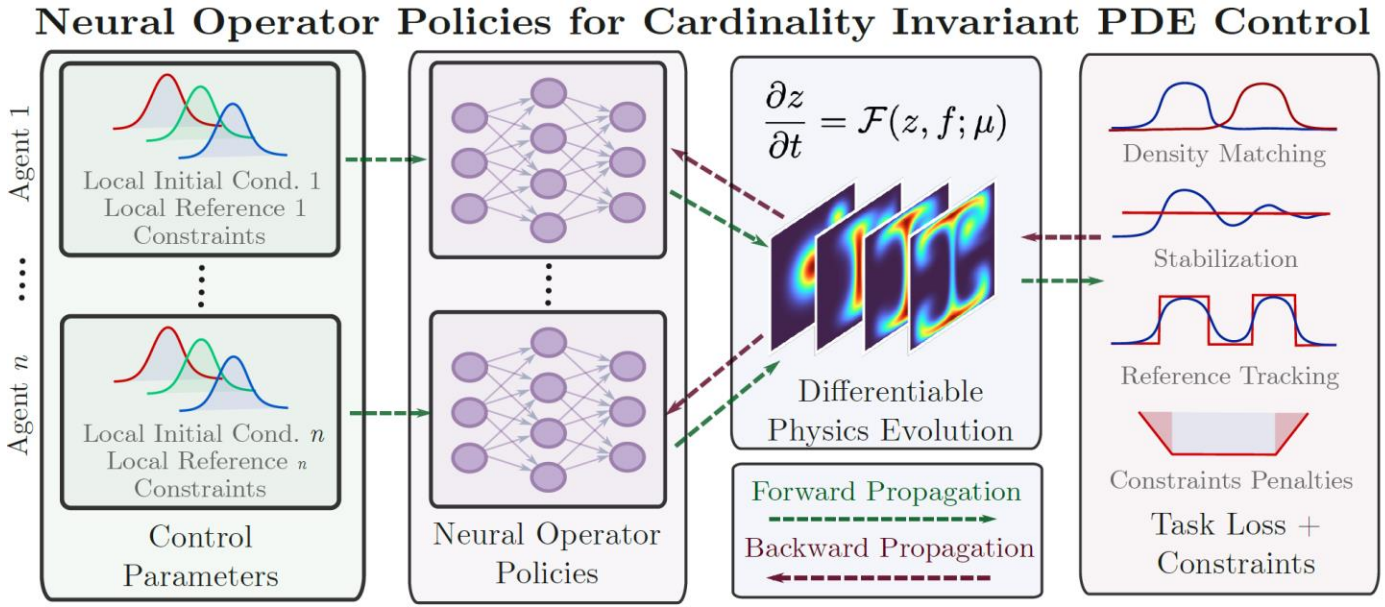


FNO



Lu, Lu, et al. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence, 2021
Nikola Kovachki, et al., Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs, JMLR, 2023

Learning to Control PDE Example in JAX



Pietro Zanotta

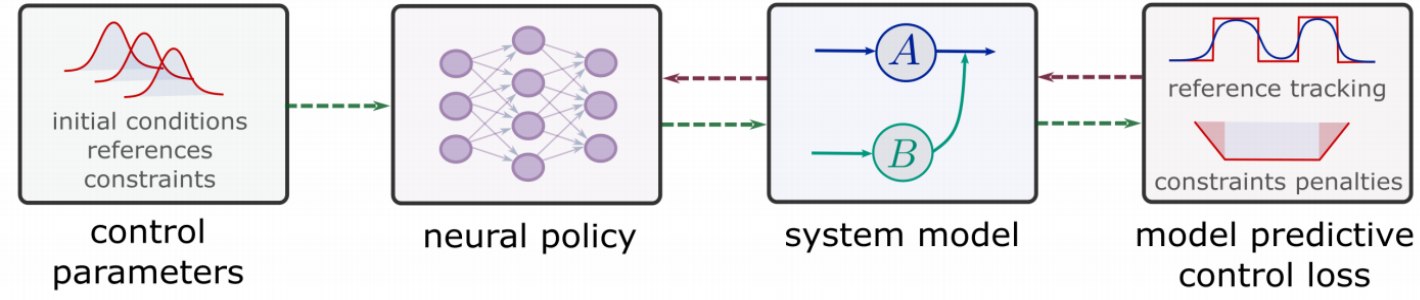
colab.research.google.com/github/SciML-L4DC2026/SciML-L4DC2026/blob/main/notebooks/L2C/Example_2_DPC_PDE/Example_2_DPC_PDE.ipynb

sciml-l4dc2026.github.io/SciML-L4DC2026

Learning to Control (L2C) Summary

- **L2C via Differentiable Predictive Control**
  - Learns approximate explicit MPC policies through sampling-based self-supervised optimization.
  - Shifts computational burden offline, enabling real-time deployment via a single policy evaluation.
  - Works with differentiable ODE/PDE solvers and a wide range of learned surrogate models.

- **Challenges**
  - Control of hyperbolic PDEs
  - Handling with stiff and hybrid systems
  - Gradients ill conditioning
  - Offline pre-training vs online adaptation
- **Code**
  - github.com/pnnl/neuromancer
  - github.com/SOLARIS-JHU/CINOC
- **Hiring!** jdrgona1@jh.edu

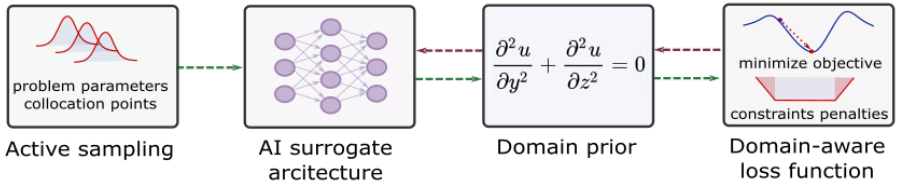


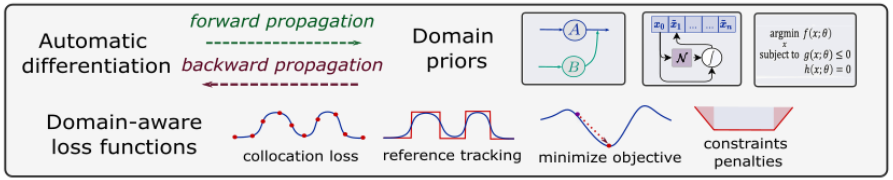
SciML Tutorial Summary

Scientific machine learning (SciML) integrates deep learning, constrained optimization, physics-based modeling, and control in a unified abstraction.

- Learning to optimize (L2O)
- Learning to model (L2M)
- Learning to control (L2C)

sciml-l4dc2026.github.io/SciML-L4DC2026/





Lab websites and contact

Thomas Beckers

- www.tbeckers.com/
- thomas.beckers@vanderbilt.edu

Truong Nghiem

- truong.nxtlab.org/
- truong.nghiem@ucf.edu

Jan Drgona

- solaris-jhu.github.io/solaris-website/
- drgona.github.io/
- github.com/drgona/SciML-Course
- jdrgona1@jh.edu

VANDERBILT UNIVERSITY

University of Central Florida

JOHNS HOPKINS UNIVERSITY