Optimal Control Operator Perspective and a Neural Adaptive Spectral Method

Mingquan Feng[1], **Zhijie Chen**[2], **Yixin Huang**[1], **Yizhou Liu**[1], **Junchi Yan**[1,3]

[1]School of Electronic Information & Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
[2]Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
[3]School of Artificial Intelligence & MoE Lab of AI, Shanghai Jiao Tong University, Shanghai, China
{fengmingquan, cindyh1103, yizhou0409, yanjunchi}@sjtu.edu.cn, lucmon@illinois.edu

Abstract

Optimal control problems (OCPs) involve finding a control function for a dynamical system such that a cost functional is optimized. It is central to physical systems in both academia and industry. In this paper, we propose a novel instance-solution control operator perspective, which solves OCPs in a one-shot manner without direct dependence on the explicit expression of dynamics or iterative optimization processes. The control operator is implemented by a new neural operator architecture named Neural Adaptive Spectral Method (NASM), a generalization of classical spectral methods. We theoretically validate the perspective and architecture by presenting the approximation error bounds of NASM for the control operator. Experiments on synthetic environments and a real-world dataset verify the effectiveness and efficiency of our approach, including substantial speedup in running time, and high-quality in- and out-of-distribution generalization.

Code — https://github.com/FengMingquan-sjtu/NASM
Extended version — https://arxiv.org/abs/2412.12469

1 Introduction

Although control theory has been rooted in a model-based design and solving paradigm, the demands of model reusability and the opacity of complex dynamical systems call for a rapprochement of modern control theory, machine learning, and optimization. Recent years have witnessed the emerging trends of control theories with successful applications to engineering and scientific research, such as robotics (Krimsky and Collins 2020), aerospace technology (He et al. 2019), and economics and management (Lapin, Zhang, and Lapin 2019) etc.

We consider the well-established formulation of optimal control (Kirk 2004) in finite time horizon $T = [t_0, t_f]$. Denote X and U as two vector-valued function sets, representing state functions and control functions respectively. Functions in X (resp. U) are defined over T and have their outputs in $\mathbb{R}^{d_x}$ (resp. $\mathbb{R}^{d_u}$). State functions $\boldsymbol{x} \in X$ and control functions $\boldsymbol{u} \in U$ are governed by a differential equation. The optimal control problem (OCP) is targeted at finding a control function that minimizes the cost functional f (Kirk 2004; Lewis, Vrabie, and Syrmos 2012):

$$\min_{\boldsymbol{u}\in U} \quad f(\boldsymbol{x}, \boldsymbol{u}) = \int_{t_0}^{t_f} p(\boldsymbol{x}(t), \boldsymbol{u}(t))\,\mathrm{d}t + h(\boldsymbol{x}(t_f)) \tag{1a}$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}}(t) = \boldsymbol{d}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \tag{1b}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0, \tag{1c}$$

where $\boldsymbol{d}$ is the dynamics of differential equations; p evaluates the cost alongside the dynamics and h evaluates the cost at the termination state $\boldsymbol{x}(t_f)$; and $\boldsymbol{x}_0$ is the initial state. We restrict our discussion to differential equation-governed optimal control problems, leaving the control problems in stochastic networks (Dai and Gluzman 2022), inventory management (Abdolazimi et al. 2021), etc. out of the scope of this paper. The analytic solution of Eq. 1 is usually unavailable, especially for complex dynamical systems. Thus, there has been a wealth of research towards accurate, efficient, and scalable numerical OCP solvers (Rao 2009) and neural network-based solvers (Kiumarsi et al. 2017). However, both classic and modern OCP solvers are facing challenges, especially in the big data era, as briefly discussed below.

1) Opacity of Dynamical Systems. Existing works (Böhme and Frank 2017a; Effati and Pakdaman 2013; Jin et al. 2020) assume the dynamical systems a priori and exploit their explicit forms to ease the optimization. However, real-world dynamical systems can be unknown and hard to model. It raises serious challenges in data collection and system identification (Ghosh, Birrell, and De Angelis 2021), where special care is required for error reduction.

2) Model Reusability. Model reusability is conceptually measured by the capability of utilizing historical data when facing an unprecedented problem instance. Since solving an individual instance of Eq. 1 from scratch is expensive, a reusable model that can be well adapted to new problems is welcomed for practical usage.

3) Running Paradigm. Numerical optimal control solvers traditionally use iterative methods before picking the control solution, thus introducing a multiplicative term regarding the iteration in the running time complexity. This sheds light on the high cost of solving a single OC problem.

4) Control Solution Continuity. Control functions are defined on a continuous domain (typically time) despite be-

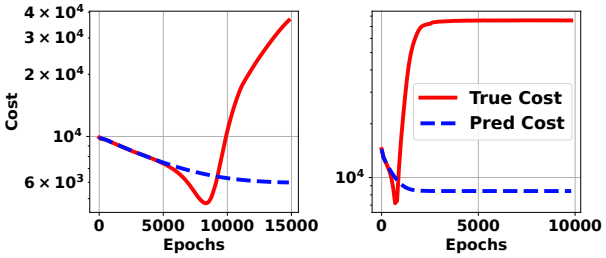


Figure 1: Phase-2 cost curves of two failed instances of two-phase control (Hwang et al. 2022) on Pendulum system. The control function gradually moves outside the training distribution of phase 1. As a result, the control function converges w.r.t. the cost predicted by the surrogate model (blue), but diverges w.r.t. true cost (red).

ing intractable for numerical solvers. Hence resorting to discretization in the input domain gives rise to the trade-off in the precision of the control solution and the computational cost, as well as the truncation errors. While the discrete solution can give point-wise queries, learning a control function for arbitrary time queries is much more valued.

5) Running Phase. The two-phase models (Chen, Shi, and Zhang 2018; Wang, Bhouri, and Perdikaris 2021; Hwang et al. 2022) can (to some extent) overcome the above issues at the cost of introducing an auxiliary dynamics inference phase. This thread of works first approximates the state dynamics by a differentiable surrogate model and then, in its second phase, solves an optimization problem for control variables. However, the two-phase paradigm increases computational cost and manifests inconsistency between the two phases. A motivating example in Fig. 1 shows that the two-phase paradigm leads to failures. When the domain of phase-2 optimization goes outside the training distribution in phase-1, it might collapse.

Table **??** compares the methods regarding the above aspects. We propose an instance-solution operator perspective for learning to solve OCPs, thereby tackling the issues above. **The highlights are:**

1) We propose the Neural Control Operator (NCO) method of optimal control, solving OCPs by learning a neural operator that directly maps problem instances to solutions. The system dynamics is implicitly learned during the training, which relies on neither any explicit form of the system nor the optimization process at test time. The operator can be reused for similar OCPs from the same distribution without retraining, unlike learning-free solvers. Furthermore, the single-phase direct mapping paradigm avoids iterative processes with substantial speedup.

2) We instantiate a novel neural operator architecture: NASM (Neural Adaptive Spectral Method) to implement the NCO and derive bounds on its approximation error.

3) Experiments on both synthetic and real systems show that NASM is versatile for various forms of OCPs. It shows over 6Kx speedup (on synthetic environments) over classical direct methods. It also generalizes well on both in- and out-of-distribution OCP instances, outperforming other neural operator architectures in most of the experiments.

Background and Related Works. Most OCPs can not be solved analytically, thus numerical methods (Körkel* et al. 2004) are developed. An OCP numerical method has three components: problem formulation, discretization scheme, and solver. The problem formulations fall into three categories: direct methods, indirect methods, and dynamic programming. Each converts OCPs into infinite-dimensional optimizations or differential equations, which are then discretized into finite-dimensional problems using schemes like finite difference, finite element, or spectral methods. These finite problems are solved by algorithms such as interior-point methods for nonlinear programming and the shooting method for boundary-value problems. These solvers rely on costly iterative algorithms and have limited ability to reuse historical data.

In addition, our work is also related to the neural operators, which are originally designed for solving parametric PDEs. Those neural operators learn a mapping from the parameters of a PDE to its solution. For example, DeepONet (DON) (Lu et al. 2021) consists of two sub-networks: a branch net for input functions (i.e. parameters of PDE) and a trunk net for the querying locations or time indices. The output is obtained by computing the inner product of the outputs of branch and trunk networks. Another type of neural operator is Fourier transform based, such as Fourier Neural Operator (FNO) (Li et al. 2020). The FNO learns parametric linear functions in the frequency domain, along with nonlinear functions in the time domain. The conversion between those two domains is realized by discrete Fourier transformation. There are many more architectures, such as Graph Element Network (GEN) (Alet et al. 2019), Spectral Neural Operator(SNO) (Fanaskov and Oseledets 2022) etc., as will be compared in our experiments.

2 Methodology

In this section, we will present the instance-solution control operator perspective for solving OCPs. The input of operator $\mathcal{G}$ is an OCP instance i, and the output is the optimal control function $\boldsymbol{u}^*$, i.e. $\mathcal{G} : I \to U$. Then we propose the Neural Control Operator (NCO), a class of end-to-end OCP neural solvers that learn the underlying infinite-dimensional operator $\mathcal{G}$ via a neural operator. A novel neural operator architecture, the Neural Adaptive Spectral Method (NASM), is implemented for the NCO method.

Instance-Solution Control Operator Perspective of OCP Solvers

This section presents a high-level instance-solution control operator perspective of OCPs. In this perspective, an OCP solver is an operator that maps problem instances (defined by cost functionals, dynamics, and initial conditions) into their solutions, i.e. the optimal controls. Denote the problem instance by $i = (f, \boldsymbol{d}, \boldsymbol{x}_{init}) \in I$, with its optimal control solution $\boldsymbol{u}^* \in U$. The operator is defined as $\mathcal{G} : I \to U$, a mapping from cost functional to optimal control.

In practice, the non-linear operator $\mathcal{G}$ can hardly have

Methods	Phase	Continuity	Dynamics	Reusability	Paradigm
Direct (Böhme and Frank 2017a)	Single	Discrete	Required	No	Iterative
Two-Phase (Hwang et al. 2022)	Two	Discrete	Dispensable[1]	Partial[2]	Iterative
PDP (Jin et al. 2020)	Single	Discrete	Required	No	Iterative
DP (Tassa, Mansard, and Todorov 2014)	Single	Discrete	Required	No	Iterative
NCO (ours)	Single	Continuous	Dispensable	Yes	One-pass

Table 1: Optimal control approaches. Our NCO covers all the merits of performing a single-phase direct-mapping paradigm without relying on known system dynamics and supports arbitrary input-domain queries.
[1] if phase-1 uses PINN loss (Wang, Bhouri, and Perdikaris 2021): required. [2] only phase-1 is reusable.

a closed-form expression. Therefore, various numerical solvers have been developed, such as direct method and indirect method, etc, as enumerated in the related works. All those numerical solvers can be regarded as approximate operators $\mathcal{N} : I \to U$ of the exact operator $\mathcal{G}$. To measure the quality of the approximated operator, we define the approximation error as the distance between the cost of the solution and the optimal cost. Let μ be the probability measure of the problem instances, assume the control dimension $d_u = 1$, and cost $f_i > 0$ (subscript i means the cost is dependent on i), w.l.o.g. Then the *approximation error* is defined as:

$$\widehat{\mathscr{E}} := \int_I \left| \frac{f_i \circ \mathcal{N}(i) - f_i \circ \mathcal{G}(i)}{f_i \circ \mathcal{G}(i)} \right| \mathrm{d}\mu(i). \tag{2}$$

Classical numerical solvers are generally guaranteed to achieve low approximation error (with intensive computing cost). For efficiency, we propose to *approximate the operator $\mathcal{G}$ by neural networks*, i.e. **Neural Control Operator (NCO)** method. The networks are trained by demonstration data pairs $(i, \boldsymbol{u}^*)$ produced by some numerical solvers, without the need for explicit knowledge of dynamics. Once trained, they infer optimal control for any unseen instances with a single forward pass. As empirically shown in our experiments, the efficiency (at inference) of neural operators is consistently better than that of classical solvers. We will also provide an approximation error bound for our specific implementation NASM of NCO.

Before introducing the network architecture, it is necessary to clarify a common component, the Encoder, used in both classical and neural control operators. The encoder converts the infinite-dimensional input i to finite-dimensional representation $\boldsymbol{e}$. Take the cost functional for example. If the cost functional $f(x, u) = \int (x - x_{\text{target}})^2 \mathrm{d}t$ is explicitly known, then the f may be encoded as the symbolic expression or the parameters x_{target} only. Otherwise, the cost functional can be represented by sampling on grids. After encoding, The encoded vector $\boldsymbol{e}$ is fed into numerical algorithms or neural networks in succeeding modules. The encoder's design is typically driven by the problem setting rather than the specific solver or network architecture.

Neural Adaptive Spectral Method

From the operator's perspective, constructing an OCP solver is equivalent to an operator-learning problem. We propose to learn the operator by neural network in a data-driven manner. Now we elaborate on a novel neural operator architecture named Neural Adaptive Spectral Method (NASM), inspired by the spectral method.

The spectral method assumes that the solution is the linear combination of p basis functions:

$$\mathcal{N}_{\text{spec}}(i)(t) = \sum_{j=1}^{p} c_j(i) b_j(t), \tag{3}$$

where the basis functions $\{b_j\}_p$ are chosen as orthogonal polynomials e.g. Fourier series (trigonometric polynomial) and Chebyshev polynomials. The coefficients $\{c_j\}_p$ are derived from instance i by a numerical algorithm. The Spectral Neural Operator (SNO) (Fanaskov and Oseledets 2022) obtains the coefficients by a network.

Our NASM model is based on a similar idea but with more flexible and adaptive components than SNO. The first difference is that summation $\sum$ is extended to aggregation $\bigoplus$. The aggregation is defined as any mapping $\mathbb{R}^p \to \mathbb{R}$ and can be implemented by either summation or another network. Secondly, the coefficient $\{c_j\}_p$ is obtained from Coefficient Net, which is dependent on not only instance i but time index t as well, inspired by time-frequency analysis (Cohen 1995). It is a generalization and refinement of $\mathcal{N}_{\text{spec}}$(Eq. 3), for the case when the characteristics of $\mathcal{G}$ are non-static. For example, $\mathcal{N}_{\text{spec}}$ with Fourier basis can only capture globally periodic functions, while the $\mathcal{N}_{\text{NASM}}$ with the same basis can model both periodic and non-periodic functions. Lastly, the basis functions $\{b_j\}_p$ are adaptive instead of fixed. The adaptiveness is realized by parameterizing basis functions by θ, which is inferred from t, i also by the neural network.

$$\mathcal{N}_{\text{NASM}}(i)(t) = \bigoplus_{j=1}^{p} c_j(t, i) b_j(t; \boldsymbol{\theta}(t, i)). \tag{4}$$

As a concrete example of adaptive basis functions, the adaptive Fourier series is obtained from the original basis by scaling and shifting, according to the parameter $\boldsymbol{\theta}$. We limit the absolute value of elements of $\boldsymbol{\theta}$ to avoid overlapping the adaptive range of basis functions. Following this principle, one can design adaptive versions for other basis function sets.

$$\begin{aligned} \{b_j(t; \boldsymbol{\theta})\} = \{1,\ \sin(\pi[(1+\theta_1)t + \theta_2]), \\ \cos(\pi[(1+\theta_3)t + \theta_4]),\ \cdots\}, \quad |\theta_k| \le 0.5, \forall k. \end{aligned} \tag{5}$$

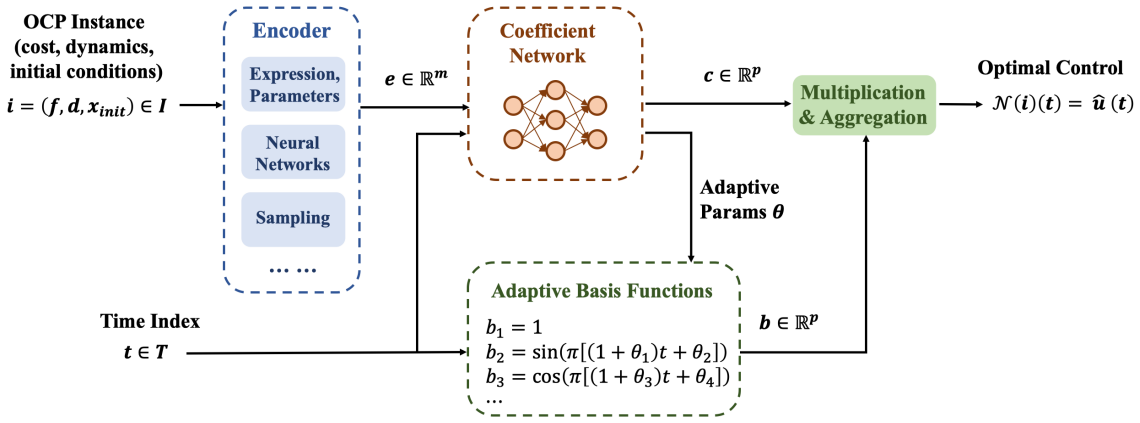


Figure 2: The architecture of NASM. The network takes two inputs: OCP instance i and time index t. The input i is pre-processed by the Encoder. Then both t and encoding $\boldsymbol{e}$ are fed into the Coefficient Network to obtain coefficients $\boldsymbol{c}$ and adaptive parameters $\boldsymbol{\theta}$. The adaptive basis (e.g. Fourier series) outputs function values $\boldsymbol{b}$, which is multiplied with $\boldsymbol{c}$ and aggregated to the final output $\hat{\boldsymbol{u}}(t)$, the estimation of optimal control for instance i at time t. Detailed explanation is given in Section 2.

The overall architecture of NASM is given in Fig. 2. The theoretic result guarantees that there exists a network instance of NASM approximating the operator $\mathcal{G}$ to arbitrary error tolerance. Furthermore, the size and depth of such a network are upper-bounded. The technical line of our analysis is partly inspired by (Lanthaler, Mishra, and Karniadakis 2022) providing error estimation for DeepONets.

Theorem 1 (**Approximator Error, Informal**). *Under some regularity assumptions, given an operator $\mathcal{G}$, and any error budget ϵ, there exists a NASM satisfying the budget with bounded network size and depth.*

Comparing NASM error bound with that of DeepONet, we discover that NASM can achieve the same error bound as DeepONet with fewer learnable parameters. More details of theorems and proofs are presented in Appx.C and Appx.D.

3 Experiments

We give experiments on synthetic and real-world environments to evaluate our instance-solution operator framework.

Synthetic Control Systems

Control Systems and Data Generation We evaluate NASM on five representative optimal control systems by following the same protocol of (Jin et al. 2020), as summarized in Table 8. We postpone the rest of the systems to Appendix E, and only describe the details of the Quadrotor control here:

$$\dot{\boldsymbol{p}} = \boldsymbol{v}, \qquad m\dot{\boldsymbol{v}} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \boldsymbol{R}^\top(\boldsymbol{q}) \begin{bmatrix} 0 \\ 0 \\ \mathbf{1}^\top \boldsymbol{u} \end{bmatrix},$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\boldsymbol{q}, \qquad \boldsymbol{J}\dot{\boldsymbol{\omega}} = \boldsymbol{T}\boldsymbol{u} - \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega}.$$

This system describes the dynamics of a helicopter with four rotors. The state $\boldsymbol{x} = [\boldsymbol{p}^\top, \boldsymbol{v}^\top, \boldsymbol{\omega}^\top]^\top \in \mathbb{R}^9$ consists of parts: position $\boldsymbol{p}$, velocity $\boldsymbol{v}$, and angular velocity $\boldsymbol{\omega}$. The control $\boldsymbol{u} \in \mathbb{R}^4$ is the thrusts of the four rotating propellers of the quadrotor. $\boldsymbol{q} \in \mathbb{R}^4$ is the unit quaternion (Jia 2019) representing the attitude (spacial rotation) of quadrotor w.r.t. the inertial frame. $\boldsymbol{J}$ is the moment of inertia in the quadrotor's frame, and $\boldsymbol{T}\boldsymbol{u}$ is the torque applied to the quadrotor. We set the initial state $\boldsymbol{x}_{init} = [[-8, -6, 9]^\top, \mathbf{0}, \mathbf{0}]^\top$, the initial quaternion $\boldsymbol{q}_{init} = \mathbf{0}$. The matrices $\boldsymbol{\Omega}(\boldsymbol{\omega}), \boldsymbol{R}(\boldsymbol{q}), \boldsymbol{T}$ are coefficient matrices, see definition in Appx. E. The cost functional is defined as $\int_0^{tf} \boldsymbol{c_x}^\top (\boldsymbol{x}(t) - \boldsymbol{x}_{goal})^2 + c_u \|\boldsymbol{u}(t)\|^2 \, \mathrm{d}t$, with coefficients $\boldsymbol{c_x} = \mathbf{1}$, $c_u = 0.1$.

In experiments presented in this section, unless otherwise specified, we temporally fix the cost functional symbolic expression, dynamics parameters, and initial condition in both training and testing. In this setting, the solution of OCP only depends on the parameter of cost, i.e. target state $\boldsymbol{x}_{goal}$. The input of NCO is the cost functional only, and the information of dynamics, and initial conditions are explicitly learned. Therefore, we generate datasets (for model training/validation) and benchmarks (for model testing) by sampling target states from a pre-defined distribution. To fully evaluate the generalization ability, we define both in-distribution (ID) and out-of-distribution (OOD) (Shen et al. 2021). Specifically, we design two random variables, $\mathbf{x}_{goal}^{in} := \boldsymbol{x}_{goal}^{base} + \boldsymbol{\epsilon}_{in}$, and $\mathbf{x}_{goal}^{out} := \boldsymbol{x}_{goal}^{base} + \boldsymbol{\epsilon}_{out}$, where $\boldsymbol{x}_{goal}^{base}$ is a baseline goal state, and $\boldsymbol{\epsilon}_{in,out}$ are different noise applied to ID and OOD. In Quadrotor problems for example, we set $\boldsymbol{x}_{goal}^{base} = \mathbf{0.6}$, and uniform noise $\boldsymbol{\epsilon}_{in} \sim \mathcal{U}(-0.5, 0.5)$, and $\boldsymbol{\epsilon}_{out} \sim \mathcal{U}(-0.7, -0.5)$.

The training data are sampled from ID, while validation data and benchmark sets are both sampled from ID and OOD separately. The data generation process is shown in Alg. 1 in Appendix. For a given distribution, we sample a target state $\boldsymbol{x}_{goal}$ and construct the corresponding cost functional f and OCP instance i. Then define 100 time indices uniformly spaced in time horizon $T = [0, \mathrm{tf}]$, $\mathrm{tf} \sim \mathcal{U}(1, 1.01)$. The length of T is slightly perturbed to add noise to the dataset.

Then we solve the resulting OCP by the Direct Method(DM) solver and get the optimal control u^* at those time indices. After that we sample 10 time indices $\{t_j\}_{1\leq j\leq 10}$, creating 10 triplets $\{(f, t_j, u^*(t_j))\}_{1\leq j\leq 10}$ and adding them to the dataset. Repeat the process, until the size meets the requirement. The benchmark set is generated in the same way, but we store (f, J_{opt}) pairs (J_{opt} is the optimal cost) instead of the optimal control. The dataset with all three factors (cost functional, dynamics parameters, and initial condition) changeable can be generated similarly.

Implementation and Baselines For all systems and all neural models, the loss is the mean squared error defined below, with a dataset D of N samples: $\mathcal{L} = \frac{1}{N}\sum_{i,j\in D} \|\mathcal{N}(i)(t_j) - \boldsymbol{u}_i^*(t_j)\|^2$., the learning rate starts from 0.01, decaying every 1,000 epochs at a rate of 0.9. The batch size is $\min(10, 000, N)$, and the optimizer is Adam (Kingma and Ba 2014). The NASM uses 11 basis functions of the Fourier series. For comparison, we choose the following baselines. Other details of implementation and baselines are recorded in Appendix F.

1) **Direct Method (DM):** a classical direct OCP solver, with finite difference discretization and Interior Point OPTimizer (IPOPT) (Biegler and Zavala 2009) backend NLP solver.

2) **Pontryagin Differentiable Programming (PDP)** (Jin et al. 2020): an adjoint-based indirect method, differentiating through PMP, and optimized by gradient descent.

3) **DeepONet (DON)** (Lu et al. 2021): The seminal work of neural operator. The DON output is a linear combination of basis function values, and both coefficients and basis are pure neural networks.

4) **Multi-layer Perceptron (MLP)**: A fully connected network implementation of the neural operator.

5) **Fourier Neural Operator (FNO)** (Li et al. 2020): A neural operator with consecutive Fourier transform layers, and fully connected layers at the beginning and the ending.

6) **Graph Element Network (GEN)** (Alet et al. 2019): Neural operator with graph convolution backbone.

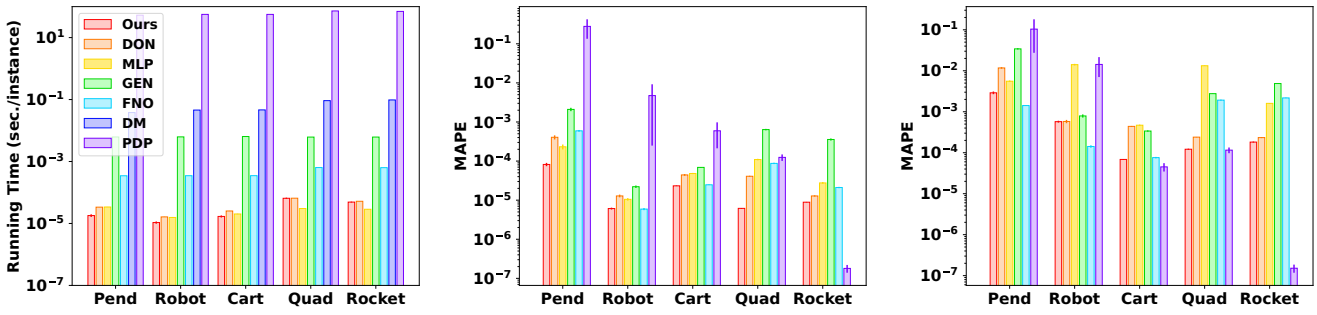
7) **Spectral Neural Operator (SNO)** (Fanaskov and Oseledets 2022): linear combination of a fixed basis (Eq. 3), with neuralized coefficients. It is a degenerated version of NASM and will be compared in the ablation study part.

Results and Discussion We present the numerical results on the five systems to evaluate the efficiency and accuracy of NASM and other architectures for instance-solution operator framework. The metrics of interest are 1) the running time of solving problems; 2) the quality of solution, measured by mean absolute percentage error (MAPE) between the true optimal cost and the predicted cost, which is defined as the mean of $|(J_{opt} - J_{sol})/J_{opt}|$, where J_{opt} is the optimal cost generated by DM (regarded as ground truth), and J_{sol} is the cost of the solution produced by the model. The MAPE is calculated on ID/OOD benchmarks respectively, and the running time is averaged for 2,000 random problems. The results of ODE-constrained OCP are visualized in Fig. 3.

First, the comparison of the wall-clock running time is shown in Fig. 3a as well as in the third column of Tab. 2, which shows that the neural operator solvers are much faster than the classic solvers, although they both tested on CPU for fairness. For example, NASM achieves over 6000 times speedup against the DM solver. The acceleration can be reasoned in two aspects: 1) the neural operator solvers produce the output by a single forward propagation, while the classic methods need to iterate between forward and backward pass multiple times; 2) the neural solver calculation is highly paralleled. Among the neural operator solvers, the running time of NASM, DON, and MLP are similar. The FNO follows a similar diagram as MLP, but 10+ times slower than MLP, since it involves computationally intensive Fourier transformations. The GEN is 100+ times slower than MLP, probably because of the intrinsic complexity of graph construction and graph convolution.

The accuracy on in- and out-of-distribution benchmark sets is compared in Fig. 3b-3c. Compared with other neural models, NASM achieves better or comparable accuracy in general. In addition, NASM outperforms classical PDP on more than half of the benchmarks. As a concrete example, we investigate the performance of the Quadrotor environment. From Tab. 2, one can observe that MAPE of NASM on the ID and OOD is the best among all neural models. The OOD performance is close to that of the classical method PDP, which is insensitive to distribution shift. We conjecture that both ID/OOD generalization ability of NASM results from its architecture, where coefficients and basis functions are explicitly disentangled. Such a structure may inherit the inductive bias from numerical basis expansion methods (e.g. (Kafash, Delavarkhalafi, and Karbassi 2014)), thus being more robust to distribution shifts.

For NASM in all synthetic environment experiments, we choose MLP as the backbone of Coefficient Net, Fourier adaptive basis, non-static coefficients, and summation aggregation. To justify these architecture choices, we perform an ablation study by changing or removing each of the modules, as displayed in Tab. 3. We use the abbreviation w.o. for 'without', and w. for 'with'. The first row denotes the unchanged model, and the second and third rows are fixing basis functions, and using static (time-independent) Coefficient Net. These two modifications slightly lower the performance in both ID and OOD. The 4th row is Fourier SNO, i.e. applying the above two modifications together on NASM, which dramatically increases the error. The reason is that SNO uses a fixed Fourier basis, and thus requires the optimal control to be globally periodic for accurate interpolation, while Quadrotor control is non-periodic. If the basis of SNO changes to non-periodic, such as Chebyshev basis as in the 5th row, then the performance is covered. The result shows that the accuracy of SNO depends heavily on the choice of pre-defined basis function. In comparison, NASM is less sensitive to the choice of basis due to its adaptive nature, as the 6th row shows that changing the basis from Fourier to Chebyshev hardly changes the NASM's performance. The 7th and 8th rows demonstrate that both replacing summation with a neural network aggregation and switching to a Convolutional network backbone do not improve the performance on Quadrotor.



(a) Time (sec./instance) (b) ID MAPE (c) OOD MAPE

Figure 3: Inference time and mean absolute percentage error (MAPE) on in-distribution (ID) and OOD benchmarks. NASM (red bars) achieves higher or comparable accuracy, with the fastest or second fastest speed.

Formulation	Model	Time(sec./instance)	ID MAPE	OOD MAPE
Direct Method	DM	9.23×10^{-2}	\	\
Indirect Method	PDP	7.25×10^{1}	1.24×10^{-4}	1.16×10^{-4}
Instance-Solution Control Operator	NASM	6.50×10^{-5}	$\mathbf{6.17 \times 10^{-6}}$	$\mathbf{1.21 \times 10^{-4}}$
	DON	6.54×10^{-5}	4.09×10^{-5}	2.40×10^{-4}
	MLP	$\mathbf{3.04 \times 10^{-5}}$	1.10×10^{-4}	1.33×10^{-2}
	GEN	6.15×10^{-3}	6.40×10^{-4}	2.77×10^{-3}
	FNO	6.38×10^{-4}	8.73×10^{-5}	1.92×10^{-3}

Table 2: Results of Quadrotor environment.

	Modification	ID MAPE	OOD MAPE
1	NASM	$\mathbf{6.17 \times 10^{-6}}$	$\mathbf{1.21 \times 10^{-4}}$
2	w.o. Adapt Params	7.86×10^{-6}	1.97×10^{-4}
3	w.o. Non-static Coef	9.01×10^{-6}	5.83×10^{-4}
4	SNO (Fourier)	9.59×10^{-2}	8.39×10^{-2}
5	SNO (Chebyshev)	6.64×10^{-6}	3.82×10^{-4}
6	w. Chebyshev	6.36×10^{-6}	2.86×10^{-4}
7	w. Neural Aggr	7.13×10^{-6}	1.29×10^{-4}
8	w. Conv	1.16×10^{-5}	4.95×10^{-3}

Table 3: Architecture justification of NASM on Quadrotor.

Extended Experiment Result on Quadrotor **1) More Variables.** In the previous experiments, only the cost function (target state) is changeable. Here, we add more variables to the network input, such that the cost function, dynamics (physical parameters e.g. wing length), and initial condition are all changeable. The in/out- distributions for dynamics and initial conditions are uniform distributions, designed similarly to that of the target state. The result is listed in Tab. 4.

2) Extreme OOD shifts and Transfer Learning. In addition to the OOD shifts in the previous experiments, we design some more extremely shifted distributions: $\boldsymbol{\epsilon}^1_{out} \sim \mathcal{U}(-1.0, -0.8), \boldsymbol{\epsilon}^2_{out} \sim \mathcal{U}(-1.3, -1.1), \boldsymbol{\epsilon}^3_{out} \sim \mathcal{U}(-1.6, -1.4)$

Model	Time(sec./instance)	ID MAPE	OOD MAPE
DM	1.59×10^{-1}	\	\
PDP	7.25×10^{1}	1.13×10^{-4}	1.15×10^{-4}
NASM	4.40×10^{-5}	$\mathbf{5.92 \times 10^{-5}}$	$\mathbf{1.57 \times 10^{-4}}$
DON	$\mathbf{3.39 \times 10^{-5}}$	1.02×10^{-4}	3.19×10^{-4}
MLP	8.45×10^{-5}	5.94×10^{-4}	2.36×10^{-3}
GEN	1.10×10^{-2}	6.06×10^{-4}	2.32×10^{-3}
FNO	1.95×10^{-3}	6.22×10^{-4}	2.32×10^{-3}

Table 4: Changeable cost, dynamics, initialization on Quad.

Model	$\boldsymbol{\epsilon}^1_{out}$	$\boldsymbol{\epsilon}^2_{out}$	$\boldsymbol{\epsilon}^3_{out}$
NASM	$\mathbf{1.29 \times 10^{-3}}$	6.90×10^{-3}	1.50×10^{-2}
DON	2.32×10^{-3}	7.58×10^{-3}	1.61×10^{-2}
MLP	3.15×10^{-3}	$\mathbf{6.10 \times 10^{-3}}$	$\mathbf{9.53 \times 10^{-3}}$
GEN	1.14×10^{-2}	3.94×10^{-2}	2.18×10^{-1}
FNO	1.35×10^{-2}	5.04×10^{-2}	1.03×10^{-1}

Table 5: MAPE of Quadrotor OOD shift, without finetune.

Tab. 5 shows the OOD MAPE on those distributions. From the table, one can observe that the performance of neural models decreases with more distribution shifts. When the distribution shift increases to $\boldsymbol{\epsilon}^3_{out}$, the results of neural mod-

Model	ϵ^1_{out}	ϵ^2_{out}	ϵ^3_{out}
NASM	$\mathbf{1.02 \times 10^{-5}}$	$\mathbf{2.29 \times 10^{-5}}$	3.07×10^{-5}
DON	1.95×10^{-5}	2.57×10^{-5}	$\mathbf{3.02 \times 10^{-5}}$
MLP	3.10×10^{-5}	1.01×10^{-4}	1.88×10^{-4}
GEN	2.36×10^{-4}	4.07×10^{-4}	8.06×10^{-4}
FNO	4.34×10^{-5}	1.16×10^{-4}	2.11×10^{-4}

Table 6: MAPE of Quadrotor OOD shift, with finetune.

els are almost unacceptable.

In practice, if large shift OOD reusability is required for some applications, we may assume a few training samples from OOD are available (i.e. few-shot). NCOs can be fine-tuned on the training samples from OOD to achieve better performance. This idea has been proposed for DeepONet transfer learning in (Goswami et al. 2022).

We follow the fine-tuning scheme proposed in (Goswami et al. 2022). We set the size of the OOD fine-tuning dataset and #epochs to be 20% of that of the ID training dataset. The learning rate is fixed at 0.001. The embedding network, basis function net, and the first two convolution blocks are all frozen during fine-tuning since we assume they keep the distribution invariant information.

Tab. 6 reveals that fine-tuning greatly improves neural model performance on extreme distribution shifts. Therefore, the availability of fine-tuning on extra data benefits the reusability of neural operators on OCP.

Real-world Dataset of Planar Pushing

We present how to learn OC of a robot arm for pushing objects of varying shapes on various planar surfaces. We use the *Pushing* dataset (Yu et al. 2016), a challenging dataset consisting of noisy, real-world data produced by an ABB IRB 120 industrial robotic arm (Fig. 4, right part).

The robot arm is controlled to push objects starting from various contact positions and 9 angles (initial state), along different trajectories (target state functions), with 11 object shapes and 4 surface materials (dynamics). The control function is represented by the force exerted on to object. Left of Fig. 4 gives an overview of input variables.

We apply NASM to learn a mapping from a pushing OCP instance (represented by variables above) to the optimal control function. The input now is no longer the parameters of cost functional f only, but parameters and representations of $(f, \boldsymbol{d}, \boldsymbol{x}_{init})$. The encoder is realized by different techniques for different inputs, such as Savitzky–Golay smoothing (Savitzky and Golay 1964) and down-sampling for target trajectories, mean and standard value extraction for friction map, and CNN for shape images.

We extract training data from ID, validation, and test data from both ID/OOD. The ID/OOD is distinguished by different initial contact positions of the arm and object. The accuracy metric MAPE is now defined as $\|\hat{u} - u^*\|/\|u^*\|$. We compare NASM performance only with neural baselines since the explicit expression of pushing OCP is unavailable, thus classical methods DM and PDP are not applicable. All neural models share the same encoder structure, while the parameters of the encoder are trained end-to-end individually for each model. The results are displayed in Tab. 7, from which one can observe that NASM outperforms all baselines in the ID MAPE and that NASM achieves the second lowest OOD MAPE, with a slight disadvantage against DON.

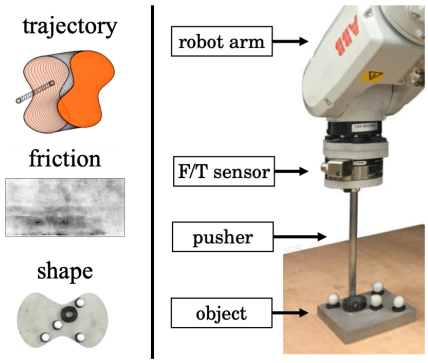


Figure 4: Pushing environment(Yu et al. 2016).

Model	ID MAPE	OOD MAPE
NASM	**0.108** ($\pm$ 0.006)	0.137 ($\pm$ 0.007)
NASM(Fourier)	0.113 ($\pm$ 0.006)	0.139 ($\pm$ 0.009)
NASM(w.o. Conv)	0.132 ($\pm$ 0.008)	0.151 ($\pm$ 0.009)
NASM(w.o. NAggr)	0.125 ($\pm$ 0.007)	0.172 ($\pm$ 0.009)
DON	0.117 ($\pm$ 0.006)	**0.134** ($\pm$ 0.008)
MLP	0.128 ($\pm$ 0.006)	0.149 ($\pm$ 0.008)
GEN	0.209 ($\pm$ 0.014)	0.199 ($\pm$ 0.015)
FNO	0.131 ($\pm$ 0.008)	0.172 ($\pm$ 0.009)

Table 7: Results of Pushing environment.

The design of NASM in the pushing environment is an adaptive Chebyshev basis, non-static Coefficient Net with Convolution backbone, and neuralized aggregation. The justification of the design is shown in Tab. 7. Its performance is insensitive to the choice of basis, as the second row reveals. Both the convolution backbone and the neuralized aggregation contribute to the high accuracy. A possible reason is that the control operator $\mathcal{G}$ in the pushing dataset is highly non-smooth and noisy, thus a more complicated architecture with higher capability and flexibility is preferred.

4 Conclusion and Outlook

We have proposed an instance-solution operator perspective of OCPs, where the operator directly maps cost functionals to OC functions. We then present a neural operator NASM, with a theoretic guarantee on its approximation capability. Experiments show outstanding generalization ability and efficiency, in both ID and OOD settings. We envision the proposed model will be beneficial in solving numerous high-dimensional problems in the learning and control fields.

We currently do not specify the forms of problem instances (e.g. ODE/PDE-constrained OCP) or investigate sophisticated models for specific problems, which calls for careful designs and exploitation of the problem structures.

Acknowledgements

This work was in part supported by the NSFC (62222607, 623B1009).

References

Abdolazimi, O.; Shishebori, D.; Goodarzian, F.; Ghasemi, P.; and Appolloni, A. 2021. Designing a new mathematical model based on ABC analysis for inventory control problem: A real case study. *RAIRO-operations research*, 55(4): 2309–2335.

Al-Tamimi, A.; Lewis, F. L.; and Abu-Khalaf, M. 2008. Discrete-time nonlinear HJB solution using approximate dynamic programming: Convergence proof. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4): 943–949.

Alet, F.; Jeewajee, A. K.; Villalonga, M. B.; Rodriguez, A.; Lozano-Perez, T.; and Kaelbling, L. 2019. Graph element networks: adaptive, structured computation and memory. In *International Conference on Machine Learning*, 212–222. PMLR.

Anandkumar, A.; Azizzadenesheli, K.; Bhattacharya, K.; Kovachki, N.; Li, Z.; Liu, B.; and Stuart, A. 2020. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

Andersson, J. A. E.; Gillis, J.; Horn, G.; Rawlings, J. B.; and Diehl, M. 2019. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1): 1–36.

Bazaraa, M. S.; Sherali, H. D.; and Shetty, C. M. 2013. *Nonlinear programming: theory and algorithms*. John Wiley & Sons.

Bellman, R.; and Kalaba, R. 1960. Dynamic programming and adaptive processes: mathematical foundation. *IRE transactions on automatic control*, (1): 5–10.

Bettiol, P.; and Bourdin, L. 2021. Pontryagin maximum principle for state constrained optimal sampled-data control problems on time scales. *ESAIM: Control, Optimisation and Calculus of Variations*, 27: 51.

Biegler, L. T.; and Zavala, V. M. 2009. Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33(3): 575–582.

Bock, H. G.; and Plitt, K.-J. 1984. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2): 1603–1608.

Boggs, P. T.; and Tolle, J. W. 1995. Sequential quadratic programming. *Acta numerica*, 4: 1–51.

Böhme, T. J.; and Frank, B. 2017a. *Direct Methods for Optimal Control*, 233–273. Cham: Springer International Publishing. ISBN 978-3-319-51317-1.

Böhme, T. J.; and Frank, B. 2017b. *Indirect Methods for Optimal Control*, 215–231. Cham: Springer International Publishing. ISBN 978-3-319-51317-1.

Chang, Y.-C.; Roohi, N.; and Gao, S. 2019. Neural lyapunov control. *Advances in neural information processing systems*, 32.

Chen, T.; and Chen, H. 1995. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4): 911–917.

Chen, Y.; Shi, Y.; and Zhang, B. 2018. Optimal control via neural networks: A convex approach. *arXiv preprint arXiv:1805.11835*.

Cheng, L.; Wang, Z.; Song, Y.; and Jiang, F. 2020. Real-time optimal control for irregular asteroid landings using deep neural networks. *Acta Astronautica*, 170: 66–79.

Cohen, L. 1995. *Time-frequency analysis*, volume 778. Prentice hall New Jersey.

Dai, J. G.; and Gluzman, M. 2022. Queueing network controls via deep reinforcement learning. *Stochastic Systems*, 12(1): 30–67.

Davis, P. J. 1975. *Interpolation and approximation*. Courier Corporation.

Dontchev, A. L.; and Zolezzi, T. 2006. *Well-posed optimization problems*. Springer.

D'ambrosio, A.; Schiassi, E.; Curti, F.; and Furfaro, R. 2021. Pontryagin neural networks with functional interpolation for optimal intercept problems. *Mathematics*, 9(9): 996.

Effati, S.; and Pakdaman, M. 2013. Optimal control problem via neural networks. *Neural Computing and Applications*, 23(7): 2093–2100.

Fanaskov, V.; and Oseledets, I. 2022. Spectral Neural Operators. *arXiv preprint arXiv:2205.10573*.

Gao, S.; Kong, S.; and Clarke, E. M. 2013. dReal: An SMT solver for nonlinear theories over the reals. In *Automated Deduction–CADE-24: 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings 24*, 208–214. Springer.

Ghosh, S.; Birrell, P.; and De Angelis, D. 2021. Variational inference for nonlinear ordinary differential equations. In *International Conference on Artificial Intelligence and Statistics*, 2719–2727. PMLR.

Goswami, S.; Kontolati, K.; Shields, M. D.; and Karniadakis, G. E. 2022. Deep transfer operator learning for partial differential equations under conditional shift. *Nature Machine Intelligence*, 1–10.

Gühring, I.; Kutyniok, G.; and Petersen, P. 2020. Error bounds for approximations with deep ReLU neural networks in W s, p norms. *Analysis and Applications*, 18(05): 803–859.

He, X.; Li, J.; Mader, C. A.; Yildirim, A.; and Martins, J. R. 2019. Robust aerodynamic shape optimization—from a circle to an airfoil. *Aerospace Science and Technology*, 87: 48–61.

Hewing, L.; Wabersich, K. P.; Menner, M.; and Zeilinger, M. N. 2020. Learning-based model predictive control: Toward safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems*, 3: 269–296.

Hwang, R.; Lee, J. Y.; Shin, J. Y.; and Hwang, H. J. 2022. Solving pde-constrained control problems using operator learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 4504–4512.

Jia, Y.-B. 2019. Quaternions. *Com S*, 477: 577.

Jin, W.; Wang, Z.; Yang, Z.; and Mou, S. 2020. Pontryagin differentiable programming: An end-to-end learning and control framework. *Advances in Neural Information Processing Systems*, 33: 7979–7992.

Kafash, B.; Delavarkhalafi, A.; and Karbassi, S. 2014. A numerical approach for solving optimal control problems using the Boubaker polynomials expansion scheme. *J. Interpolat. Approx. Sci. Comput*, 3: 1–18.

Khoo, Y.; Lu, J.; and Ying, L. 2021. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3): 421–435.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kirk, D. E. 2004. *Optimal control theory: an introduction*. Courier Corporation.

Kiumarsi, B.; Vamvoudakis, K. G.; Modares, H.; and Lewis, F. L. 2017. Optimal and autonomous control using reinforcement learning: A survey. *IEEE transactions on neural networks and learning systems*, 29(6): 2042–2062.

Körkel*, S.; Kostina, E.; Bock, H. G.; and Schlöder, J. P. 2004. Numerical methods for optimal control problems in design of robust optimal experiments for nonlinear dynamic processes. *Optimization Methods and Software*, 19(3-4): 327–338.

Krimsky, E.; and Collins, S. H. 2020. Optimal control of an energy-recycling actuator for mobile robotics applications. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 3559–3565. IEEE.

Lanthaler, S.; Mishra, S.; and Karniadakis, G. E. 2022. Error estimates for deeponets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1): tnac001.

Lapin, A.; Zhang, S.; and Lapin, S. 2019. Numerical solution of a parabolic optimal control problem arising in economics and management. *Applied Mathematics and Computation*, 361: 715–729.

Lasota, A. 1968. A discrete boundary value problem. In *Annales Polonici Mathematici*, volume 20, 183–190. Institute of Mathematics Polish Academy of Sciences.

Lewis, F. L.; Vrabie, D.; and Syrmos, V. L. 2012. *Optimal control*. John Wiley & Sons.

Li, W.; and Todorov, E. 2004. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, 222–229. Citeseer.

Li, Z.; Kovachki, N. B.; Azizzadenesheli, K.; Bhattacharya, K.; Stuart, A.; Anandkumar, A.; et al. 2020. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*.

Lu, L.; Jin, P.; Pang, G.; Zhang, Z.; and Karniadakis, G. E. 2021. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3): 218–229.

Mathiesen, F. B.; Yang, B.; and Hu, J. 2022. Hyperverlet: A Symplectic Hypersolver for Hamiltonian Systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 4575–4582.

Mehrotra, S. 1992. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4): 575–601.

Nubert, J.; Köhler, J.; Berenz, V.; Allgöwer, F.; and Trimpe, S. 2020. Safe and fast tracking on a robot manipulator: Robust mpc and neural network control. *IEEE Robotics and Automation Letters*, 5(2): 3050–3057.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*.

Petersen, P.; and Voigtlaender, F. 2020. Equivalence of approximation by convolutional neural networks and fully-connected networks. *Proceedings of the American Mathematical Society*, 148(4): 1567–1581.

Pontryagin, L. S. 1987. *Mathematical theory of optimal processes*. CRC press.

Powell, M. J. D.; et al. 1981. *Approximation theory and methods*. Cambridge university press.

Raissi, M.; Perdikaris, P.; and Karniadakis, G. E. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378: 686–707.

Raissi, M.; Yazdani, A.; and Karniadakis, G. E. 2020. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481): 1026–1030.

Rao, A. V. 2009. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1): 497–528.

Savitzky, A.; and Golay, M. J. 1964. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8): 1627–1639.

Shen, Z.; Liu, J.; He, Y.; Zhang, X.; Xu, R.; Yu, H.; and Cui, P. 2021. Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*.

Spong, M. W.; Hutchinson, S.; and Vidyasagar, M. 2020. *Robot modeling and control*. John Wiley & Sons.

Tassa, Y.; Mansard, N.; and Todorov, E. 2014. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 1168–1175. IEEE.

Tedrake, R. 2022. *Underactuated Robotics*.

Truesdell, C. A. 1992. *A First Course in Rational Continuum Mechanics V1*. Academic Press.

Wang, S.; Bhouri, M. A.; and Perdikaris, P. 2021. Fast PDE-constrained optimization via self-supervised operator learning. *arXiv preprint arXiv:2110.13297*.

Wang, S.; Teng, Y.; and Perdikaris, P. 2021. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5): A3055–A3081.

Wang, S.; Wang, H.; and Perdikaris, P. 2021. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40): eabi8605.

Xie, S.; Hu, X.; Qi, S.; and Lang, K. 2018. An artificial neural network-enhanced energy management strategy for plug-in hybrid electric vehicles. *Energy*, 163: 837–848.

Xiu, D.; and Hesthaven, J. S. 2005. High-order collocation methods for differential equations with random inputs. *SIAM Journal on Scientific Computing*, 27(3): 1118–1139.

Yu, B.; et al. 2018. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1): 1–12.

Yu, K.; Bauza, M.; Fazeli, N.; and Rodriguez, A. 2016. More than a Million Ways to be Pushed. *A High-Fidelity Experimental Data Set of Planar Pushing In: IEEE/RSJ IROS*.

Zhou, D.-X. 2020. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2): 787–794.

Zhou, R.; Quartz, T.; De Sterck, H.; and Liu, J. 2022. Neural Lyapunov Control of Unknown Nonlinear Systems with Stability Guarantees. *arXiv preprint arXiv:2206.01913*.

Zhu, Y.; and Zabaras, N. 2018. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366: 415–447.

Zhu, Y.; Zabaras, N.; Koutsourelakis, P.-S.; and Perdikaris, P. 2019. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394: 56–81.