NONLINEAR MODEL REDUCTION FOR OPERATOR LEARNING

Hamidreza Eivazi, Stefan Wittek & Andreas Rausch
ISSE, Technical University of Clausthal, 38678 Clausthal-Zellerfeld, DE
{he76,stefan.wittek,andreas.rausch}@tu-clausthal.de

ABSTRACT

Operator learning provides methods to approximate mappings between infinite-dimensional function spaces. Deep operator networks (DeepONets) are a notable architecture in this field. Recently, an extension of DeepONet based on model reduction and neural networks, proper orthogonal decomposition (POD)-DeepONet, has been able to outperform other architectures in terms of accuracy for several benchmark tests. We extend this idea towards nonlinear model order reduction by proposing an efficient framework that combines neural networks with kernel principal component analysis (KPCA) for operator learning. Our results demonstrate the superior performance of KPCA-DeepONet over POD-DeepONet.

1 INTRODUCTION

Operator learning. Partial Differential Equations (PDEs) are a fundamental mathematical tool for describing and analyzing physical phenomena that evolve in time and space (Brunton & Kutz, 2023). Solving the so-called parametric PDEs requires repeated operation of an expensive forward model (e.g. finite element methods (Hughes, 1989)) for every instance of the PDE, which demands a formidable cost. Recently, a new branch of ML research (the so-called operator learning) has made substantial advances for solving parametric PDEs by providing methods for learning operators, i.e. maps between infinite-dimensional spaces (Kovachki et al., 2023). Operator networks are, by construction, resolution-independent; the model can provide solutions for any arbitrary input coordinate. DeepONet (Lu et al., 2019; 2021) and its POD-based extension (POD-DeepONet) (Lu et al., 2022), Fourier neural operator (FNO) (Li et al., 2020), and PCA-based neural networks (PCANN) (Bhattacharya et al., 2020) are among successful operator learning approaches.

Our contributions. POD-DeepONet proposed by Lu et al. (2022) follows the idea of PCANN (Bhattacharya et al., 2020) for employing POD to represent functions. In POD-DeepONet, the trunk network of the DeepONet is replaced by a set of pre-computed POD bases obtained from the training data. However, POD is a linear decomposition technique and its ability to represent functions may be limited, especially for complex high-dimensional functions. In this contribution, we introduce kernel-PCA DeepONet (KPCA-DeepONet) as an efficient framework to combine nonlinear model reduction with operator learning.

- The KPCA-DeepONet is the first work that benefits from kernel methods and nonlinear model reduction techniques for learning operators.
- KPCA-DeepONet provides a non-linear reconstruction using kernel ridge regression.
- Our method provides less than 1% error, the lowest error reported in the literature, for the benchmark test case of the Navier–Stokes equation.

2 METHODOLOGY

Let us consider $\mathcal{U}$ and $\mathcal{V}$ as two separable Banach spaces and assume that $\mathcal{G} : \mathcal{U} \mapsto \mathcal{V}$ is an arbitrary (possibly nonlinear) operator. We consider a setting in which we only have access to partially observed input/output data $\{u_i, v_i\}_{i=1}^N$ as N elements of $\mathcal{U} \times \mathcal{V}$ such that

$$\mathcal{G}(u_i) = v_i, \qquad \text{for } i = 1, \cdots, N. \tag{1}$$

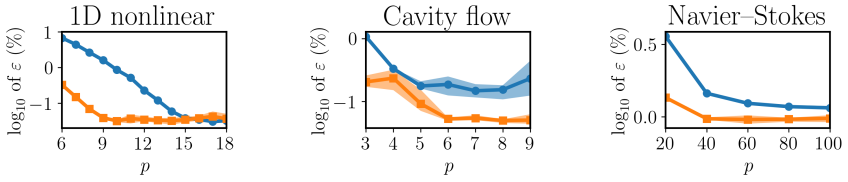

Figure 1: Comparison of the proposed KPCA-DeepONet (orange, ■) and POD-DeepONet (blue, ●). Lines and shades indicate mean and standard deviation, respectively, over 5 independent trials.

The input function u is defined on the domain $D \subset \mathbb{R}^q$ and the output function v is defined on the domain $D' \subset \mathbb{R}^{q'}$. Moreover, we consider $\mathcal{P}$ and $\mathcal{Q}$ as two linear and bounded evaluation operators such that

$$\mathcal{P} : u \mapsto (u(\boldsymbol{x}_1), u(\boldsymbol{x}_2), \cdots, u(\boldsymbol{x}_n))^T \quad \text{and} \quad \mathcal{Q} : v \mapsto (v(\boldsymbol{y}_1), v(\boldsymbol{y}_2), \cdots, v(\boldsymbol{y}_m))^T, \tag{2}$$

where $u(\boldsymbol{x}_i) \in \mathbb{R}$, $v(\boldsymbol{y}_i) \in \mathbb{R}$, and $\{\boldsymbol{x}_i\}_{i=1}^n$ and $\{\boldsymbol{y}_i\}_{i=1}^m$ indicate two sets of collocation points in the domains D and D', respectively. Considering $U_i = \mathcal{P}(u_i)$ and $V_i = \mathcal{Q}(v_i)$, our goal is to learn an approximation of $\mathcal{G}$ from the training dataset $\{U_i, V_i\}_{i=1}^N$. We refer to appendix A.1 and Lu et al. (2022) for details on DeepONet and POD-DeepONet.

KPCA-DeepONet. A diagram of our method is depicted in appendix A.2 figure 2. Let k_v, k_z be kernel functions. The KPCA bases of the output function v are computed by performing eigendecomposition on $\boldsymbol{K}_v = k_v(V_i, V_j)$ obtained from the training data. We denote the coefficients of the first p KPCA basis of v_i as $\boldsymbol{z}_i \in \mathbb{R}^p$. The reconstruction of v_i from $\boldsymbol{z}_i$ is obtained through a kernel ridge (ℓ_2-regularized) regression $h : \boldsymbol{z} \mapsto v$ following the representer theorem (appendix A.3) for achieving an optimal solution (Schölkopf & Smola, 2018). The branch network learns the mapping from the input function U_i to the coefficients of the KPCA basis $\boldsymbol{z}_i$ from data. Thus, the output of KPCA-DeepONet can be written as

$$\mathcal{G}(u)(\boldsymbol{y}) \approx \sum_{i=1}^{N} \alpha_i(\boldsymbol{y})\, k_z(\, \Big\|_{k=1}^{p} b_k(U), \boldsymbol{z}_i^t) + \phi_0(\boldsymbol{y}), \tag{3}$$

where N denotes the number of training samples, $\{b_1, b_2, \cdots, b_p\}$ are the p outputs of the branch net, $\boldsymbol{z}_i^t$ is the projection of the i-th output function V_i of the training data on the p KPCA bases, and ϕ_0 is the mean function. α_i are the weights of the kernel ridge regression. $\|$ indicates concatenation. Similar to ideas in POD-DeepONet and PCANN, we interpolate the coefficients of the kernel ridge regression to obtain $\alpha_i(\boldsymbol{y})$ and satisfy the discretization-invariance of the output function. The KPCA bases are only required for training.

3 Numerical Experiments

We compare the proposed KPCA-DeepONet with POD-DeepONet on a 1D nonlinear parametrized function taken from (Chaturantabut & Sorensen, 2010), the regularized cavity flow (Lu et al., 2022), and the Navier–Stokes equation (Lu et al., 2022). We evaluate the performance of the networks by computing the ℓ_2 relative error ε of the predictions; we perform five independent training trials to compute the mean error and the standard deviation for each test. For POD-DeepONet, we rescale the output as suggested by Lu et al. (2022). Figure 1 summarizes the results for different sizes of the latent space p, showing the superior performance of KPCA-DeepONet. For a more detailed comparison and discussion on computational cost, we refer to appendices A.4 and A.7, respectively.

4 Conclusions

Our results show that employing kernel methods and nonlinear model reduction techniques, i.e. KPCA and kernel ridge regression, combined with DeepONet can provide a more accurate framework for learning operators. The kernel ridge regression employed in KPCA-DeepONet for the reconstruction of the output function can be performed efficiently due to the low dimensionality of the problem in the latent space. Since the reconstruction is nonlinear, our method can be extended for operator learning of PDEs with discontinuities in the future.

ACKNOWLEDGEMENTS

All the codes employed for developing KPCA-DeepONet are released as open-source on GitHub-repository https://github.com/HamidrezaEiv/KPCA-DeepONet. Hamidreza Eivazi's research was conducted within the Research Training Group CircularLIB, supported by the Ministry of Science and Culture of Lower Saxony with funds from the program zukunft.niedersachsen of the Volkswagen Foundation.

URM STATEMENT

The authors acknowledge that at least one key author of this work meets the URM criteria of ICLR 2024 Tiny Papers Track.

REFERENCES

Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric PDEs. *arXiv preprint arXiv:2005.03180*, 2020. doi: 10.48550/arXiv.2005.03180.

Steven L. Brunton and J. Nathan Kutz. Machine learning for partial differential equations. *arXiv preprint arXiv:2303.17078*, 2023. doi: 10.48550/arXiv.2303.17078.

Saifon Chaturantabut and Danny C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010. doi: 10.1137/090766498.

Thomas J. R. Hughes. The finite element method: Linear static and dynamic finite element analysis. *Computer-Aided Civil and Infrastructure Engineering*, 4(3):245–246, 1989. doi: https://doi.org/10.1111/j.1467-8667.1989.tb00025.x.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2023. doi: 10.48550/arXiv.2108.08481.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020. doi: https://doi.org/10.48550/arXiv.2010.08895.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. doi: 10.48550/arXiv.1711.05101.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019. doi: 10.48550/arXiv.1910.03193.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021. doi: 10.1038/s42256-021-00302-5.

Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022. doi: https://doi.org/10.1016/j.cma.2022.114778.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 06 2018. ISBN 9780262256933. doi: 10.7551/mitpress/4175.001.0001.

A APPENDIX

A.1 DEEP OPERATOR NETWORKS (DEEPONETS)

Let us consider a stacked DeepONet with bias (Lu et al., 2019). A DeepONet consists of two sub-networks, i.e., a trunk network and a branch network. The trunk net takes the coordinates as the input and the branch net takes a discretized function U as the input. The operator $\mathcal{G}$ that maps the input function U to the output function v can be approximated as

$$\mathcal{G}(u)(\boldsymbol{y}) \approx \sum_{k=1}^{p} b_k(U) t_k(\boldsymbol{y}) + b_0, \tag{4}$$

for any point $\boldsymbol{y}$ in D', where $b_0 \in \mathbb{R}$ indicates a bias, $\{b_1, b_2, \cdots, b_p\}$ are the p outputs of the branch net, and $\{t_1, t_2, \cdots, t_p\}$ are the p outputs of the trunk net. The trunk net automatically learns a set of bases for the output function v from the training data. In POD-DeepONet (Lu et al., 2022), the trunk net is replaced by a set of POD bases, and the branch net learns their coefficients. Thus, the output can be written as

$$\mathcal{G}(u)(\boldsymbol{y}) \approx \sum_{k=1}^{p} b_k(U) \phi_k(\boldsymbol{y}) + \phi_0(\boldsymbol{y}), \tag{5}$$

where $\{\phi_1, \phi_2, \cdots, \phi_p\}$ are the POD bases of v and ϕ_0 is the mean function.

A.2 KPCA-DEEPONET DIAGRAM

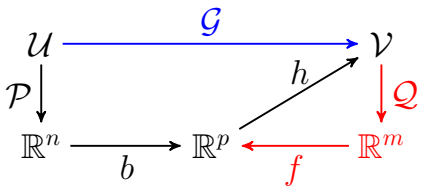

Figure 2: A diagram of the KPCA-DeepONet operator learning setup summarizing various maps of interest. $\mathcal{G}$ is the operator we want to learn. $\mathcal{P}$ and $\mathcal{Q}$ are the evaluation operators, b is the mapping by the branch network, f is the projection on the KPCA basis, and h is the mapping by the kernel ridge regression. The red color indicates those mappings that are only required for training.

A.3 THE REPRESENTER THEOREM.

Theorem A.1 *Representer theorem (Schölkopf & Smola, 2018): Let $\Omega : [0, +\infty) \mapsto \mathbb{R}$ be strictly increasing and let L be a loss function. Consider the optimization problem*

$$\min_{f \in \mathcal{F}} L(\boldsymbol{z}_i, V_i, f(\boldsymbol{z}_i)) + \lambda \Omega(||f||^2_{\mathcal{F}}), \tag{6}$$

where $\mathcal{F}$ is a reproducing kernel Hilbert spaces (RKHS) with kernel k_z, and $\lambda > 0$. Then, any optimal solution has the form of $h(\cdot) = \sum_{i=1}^{N} \alpha_i k_z(\cdot, \boldsymbol{z}_i)$, where α_i are the data-dependent weights.

A.4 FURTHER EXPERIMENTAL RESULTS

In this section, we present further results obtained from our experiments and compare them to the results reported in Lu et al. (2022). The ℓ_2 relative errors obtained from the best models are reported in table 1. Results from Lu et al. (2022) correspond to the best model or extension of a model.

Figures 3 and 4 illustrate the reference data, the prediction of KPCA-DeepONet, and the absolute error of the prediction for the Navier–Stokes and cavity flow problems, respectively. $\tilde{\cdot}$ indicates the KPCA-DeepONet prediction. Note that for the cavity flow problem, the operator network outputs two functions corresponding to the velocity in x and y directions, indicated by v_x and v_y, respectively, in figure 4.

Table 1: The ℓ_2 relative errors ε obtained from different operator networks. Results for models marked with * are taken from Lu et al. (2022).

Models	1D nonlinear	Cavity flow	Navier–Stokes
KPCA-DeepONet	$\mathbf{0.03 \pm 0.00}\%$	$\mathbf{0.05 \pm 0.00}\%$	$\mathbf{0.96 \pm 0.05}\%$
POD-DeepONet	$0.03 \pm 0.00\%$	$0.15 \pm 0.03\%$	$1.15 \pm 0.02\%$
POD-DeepONet*	–	$0.33 \pm 0.08\%$	$1.36 \pm 0.03\%$
DeepONet*	–	$1.20 \pm 0.23\%$	$1.78 \pm 0.02\%$
FNO*	–	$0.63 \pm 0.04\%$	$1.81 \pm 0.02\%$

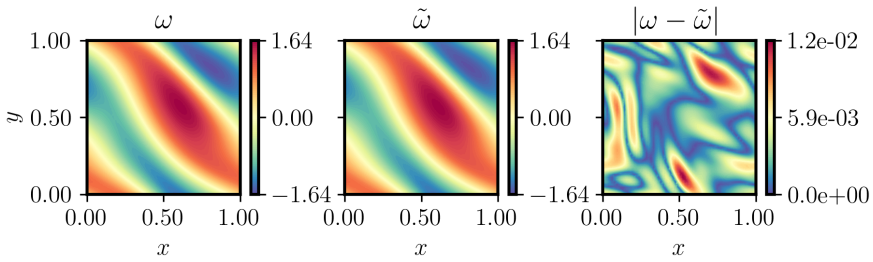

Figure 3: KPCA-DeepONet prediction against the reference data for one sample of the test dataset for the Navier–Stokes equation. $\tilde{\cdot}$ indicates the KPCA-DeepONet prediction.

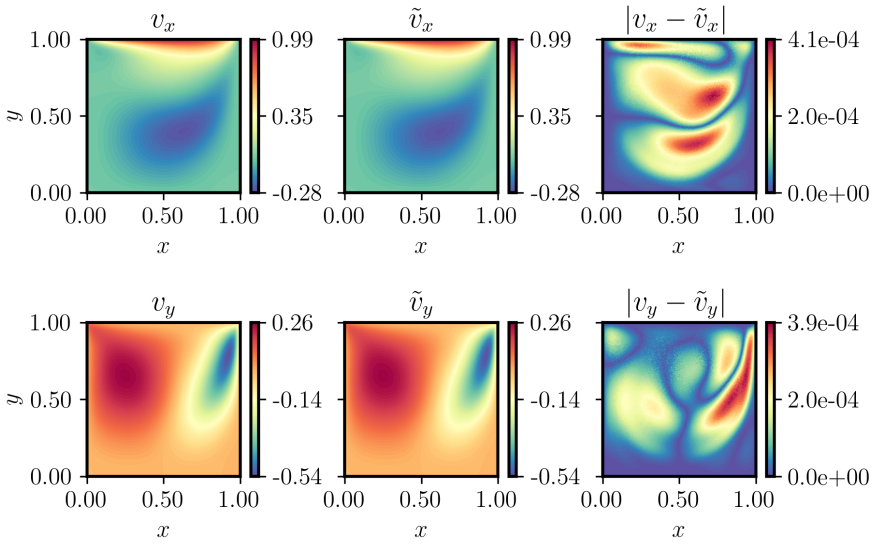

Figure 4: KPCA-DeepONet prediction against the reference data for one sample of the test dataset for the cavity flow. $\tilde{\cdot}$ indicates the KPCA-DeepONet prediction.

A.5 Test setup

The data size of each problem is reported in table 2. We refer readers to section 5.6. of Lu et al. (2022) for a detailed description of the problem setup for the Navier–Stokes equation in the vorticity-velocity form, section 5.7. of Lu et al. (2022) for the regularized cavity flow (steady) problem, and section 3.3.1. of Chaturantabut & Sorensen (2010) for the 1D nonlinear parametrized function. Note that for the 1D nonlinear problem, the goal is to learn a resolution-independent approximation of a parametrized function. The parameters of the function are the inputs of the branch network.

Table 2: Dataset size for each problem.

	No. of training data	No. of testing data
1D nonlinear	51	51
Cavity flow	100	10
Navier–Stokes	1000	200

A.6 ARCHITECTURE AND HYPERPARAMETERS

In this section, we provide details on the selected types of kernels and their hyperparameters. We also report the architecture of the branch network.

Table 3: The selected kernel parameters for each problem.

	γ_v	c_v	d_v	γ_z	c_z	d_z	λ
1D nonlinear	1.0	0.0	1	1.0	0.0	2	10^{-3}
Cavity flow	1.0	1.0	1	0.01	1.0	2	10^{-6}
Navier–Stokes	1.0	0.0	1	10^{-3}	0.1	2	10^{-3}

Kernels. Let k_v and k_z be the kernel functions for KPCA (map from the original space to latent space) and kernel ridge regression (map from the latent space to original space), respectively. For the conducted experiments, we utilize polynomial kernels as

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\gamma\, \boldsymbol{x}^T \boldsymbol{y} + c)^d, \tag{7}$$

for simplicity. Alternative kernels, including RBF, Laplace, or Matérn, can also be employed in the proposed context. Note that, in PCA, the reconstruction map involves a direct linear combination of the PCA bases. However, in kernel PCA, the reconstruction map requires a separate step, such as kernel regression, to map the reduced-dimensional representation back to the original input space. The selected kernel parameters and the regularization coefficient for kernel ridge regression λ are reported in table 3 for each problem. Note that we chose a linear kernel for mapping to the latent space in all test cases to underscore the impact of the nonlinear reconstruction on the performance of the learned operator. We compare the performance of KPCA-DeepONet when utilizing both a linear ($d_v = 1$) and a quadratic ($d_v = 2$) kernel for mapping to the latent space for the 1D nonlinear problem. The mapping from the latent space to the output function utilizes a quadratic kernel in both cases. Results are depicted in figure 5 for different sizes of the latent space. The results indicate that incorporating nonlinearity could result in better performance when an appropriate size is chosen for the latent space. We obtained ℓ_2 relative error ε of $0.02 \pm 0.00\%$ using the quadratic kernel against $0.03 \pm 0.00\%$ using the linear kernel from the best models.

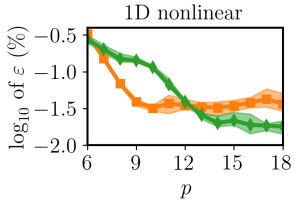

Figure 5: Performance of KPCA-DeepONet when utilizing a linear (orange, ■) and a quadratic (green, ♦) kernel for mapping to the latent space for the 1D nonlinear problem. Lines and shades indicate mean and standard deviation, respectively, over 5 independent trials.

Neural network architecture. In both KPCA-DeepONet and POD-DeepONet the mapping from the input function to the latent space is performed via the branch network. For both methods, we implement the same architectures and training processes. The branch network architecture for each

problem is reported in table 4. Both the information regarding the output function and its projection into the latent space can be used for training the branch network, with the latter being implemented in this study. The mean-squared error loss for training of the branch network can be expressed as

$$\mathcal{L}_b = \frac{1}{N_b} \sum_{i=1}^{N_b} \left\| \boldsymbol{b}_i - \boldsymbol{z}_i \right\|_2^2, \tag{8}$$

where $\boldsymbol{b}_i$ is the output of the branch network for ith input sample, $\boldsymbol{z}_i$ is the projection of the ith output function into the latent space using KPCA, and N_b is the batch size. $||\cdot||_2$ indicates ℓ_2-norm.

Table 4: Architecture of the branch network for KPCA-DeepONet and POD-DeepONet for each problem.

	Branch network	Activation function
1D nonlinear	Depth 4 & Width 64	tanh
Cavity flow	Depth 4 & Width 64	tanh
Navier–Stokes	CNN	tanh

Training. For the 1D nonlinear and cavity flow test cases the Adam algorithm is utilized as the optimizer in the training process of the neural network. For the Navier–Stokes problem we employ the AdamW algorithm (Loshchilov & Hutter, 2019). In all the cases, a scheduled learning rate is used based on the inverse time decay schedule.

A.7 Computational complexity and cost

In this section, we discuss the computational complexity and cost of the forward maps in KPCA-DeepONet and POD-DeepONet. Both approaches utilize the branch network to map the input function U to the latent vector $\boldsymbol{z}$ with the size of p. We exclude this step and only discuss the computational complexity of the reconstruction maps from the latent vector to the output function v for one sample. For KPCA-DeepONet the reconstruction map comprises two steps: (1) computing the kernel matrix k_z with the computational complexity of $\mathcal{O}(p \times N)$ for a naive implementation, where N is the number of training samples, and (2) mapping to the output function space with the computational complexity of $\mathcal{O}(N \times m)$, where m is the number of evaluation coordinates. Since $m > p$, we can conclude that the computational complexity of the reconstruction map of the proposed KPCA-DeepONet, for a naive implementation, is $\mathcal{O}(N \times m)$ while for POD-DeepONet it is $\mathcal{O}(p \times m)$. The computational complexity of the reconstruction map in KPCA-DeepONet increases linearly with the number of training samples, potentially resulting in memory challenges when dealing with large datasets. Sparse kernel methods may be a suitable remedy for this limitation.

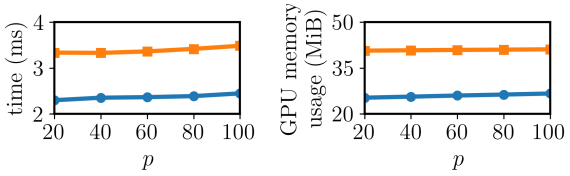

Figure 6: Computational time in milliseconds (ms) (left) and GPU memory usage (right) of the proposed KPCA-DeepONet (orange, ■) and POD-DeepONet (blue, ●) versus the size of the latent space. Results are reported for the Navier–Stokes problem.

The computational complexity for modern GPU-based implementations is notably lower than the aforementioned values. We report the computational time and GPU memory usage of the forward map of a batch of 100 samples for the Navier–Stokes problem. Results are depicted for different sizes of the latent space p in figure 6. It can be noted that the computational time and GPU memory usage of the KPCA-DeepONet are only marginally higher than those of the POD-DeepONet and do not exhibit scaling with N/p. All computations are performed on a workstation with one NVIDIA GeForce RTX 3090 GPU.