Learning solution operators of PDEs with sparse approximation methods

Sebastian Neumayer Daniel Potts Fabian Taubert

June 5, 2026

We investigate the approximation of solution operators for partial differential equations (PDEs) using sparse high-dimensional techniques. Building on a dimension-incremental framework, we combine product basis expansions with sparse recovery methods, specifically orthogonal matching pursuit (OMP), to substantially reduce the required sample size compared with a previously considered cubature-based approach. We evaluate the resulting method numerically on several examples, comparing it against both cubature-based sparse approximation and Fourier neural operators in terms of accuracy, runtime, and sample size. The experiments show that our approach considerably reduces the number of required PDE solves relative to its predecessor while maintaining competitive accuracy, particularly when the solution admits a sparse representation in the chosen basis. Furthermore, the recovered sparse index sets yield interpretable insights into the relevant variables and parameter interactions.

Keywords: dimension-incremental algorithms, nonlinear approximation, operator learning, partial differential equations, sparse recovery

2020 AMS Mathematics Subject Classification: 41A50, 42B05, 65D15, 65D40, 65T60

1 Introduction

Parameter-dependent partial differential equations (PDEs) arise naturally across a broad spectrum of scientific and engineering applications, including forward simulations, control theory, uncertainty quantification, and inverse problems. There, the repeated numerical solution of the underlying PDEs often represents the dominant computational cost, particularly for high-dimensional parameter spaces. Consequently, the efficient approximation of the solution operator, which maps parameter inputs such as initial values or forcing terms to the PDE solution, has emerged as a central problem in numerical analysis and scientific computing.

Recent studies indicate that learned approaches, see [10, 31, 48] for an overview, can outperform classical techniques [1]. As the overall computational effort is often dominated by the number of required PDE solves for generating the training samples rather than by the training itself, methods should be sample efficient. To formalize operator learning in the PDE context, let $\mathcal{F}$ denote a (potentially infinite-dimensional) parameter space, $\mathcal{U}$ a target space of functions on some domain $\Omega \subset \mathbb{R}^d$, and $G\colon \mathcal{F} \to \mathcal{U}$ the unknown solution operator. As

in reduced-order modeling [8] and consistent with the finite capacity of the hidden representations used by neural operators [29], we project the input parameters onto a finite set of basis coefficients $\boldsymbol{a} \in \mathbb{R}^n$, yielding a finite-dimensional operator $G_n \colon \mathbb{R}^n \to \mathcal{U}$. Given training data $\{(\boldsymbol{a}_i, \boldsymbol{x}_i, u_{\boldsymbol{a}_i}(\boldsymbol{x}_i))\}_{i=1}^M \subset \mathbb{R}^n \times \Omega \times \mathbb{R}$, consisting of parameter vectors paired with solution evaluations (often with multiple spatial evaluations per parameter $\boldsymbol{a}$), the goal is to learn an accurate approximation of G_n. A central difficulty is the curse of dimensionality: the sample and computational complexity may scale exponentially in the input dimension n.

To construct a tractable approximation of G_n, one must first choose a suitable parametric architecture. In approximation theory, a standard approach for handling the infinite-dimensional range space $\mathcal{U}$ is to choose a basis $\{\Phi_j\}_{j\in J} \subset \mathcal{U}$, such as the Fourier or Chebyshev basis, and to work with the truncated expansion

$$G_n[\boldsymbol{a}](\boldsymbol{x}) \approx \sum_{j\in J} b_j(\boldsymbol{a})\Phi_j(\boldsymbol{x}) \qquad \forall\, \boldsymbol{x} \in \Omega. \tag{1.1}$$

This expansion is also used by the DeepONet approach [38]. However, we restrict (1.1) to a fixed, non-learnable basis. Then, parameterizing $b \colon \mathbb{R}^n \to \mathbb{R}^{|J|}$ by a neural network $b(\cdot\,; \boldsymbol{c})$ with parameters $\boldsymbol{c} \in \mathbb{R}^P$, as in DeepONet's branch network, leads to the training problem

$$\operatorname*{arg\,min}_{\boldsymbol{c}\in\mathbb{R}^P} \sum_{i=1}^M \left| \sum_{j\in J} b_j(\boldsymbol{a}_i; \boldsymbol{c})\Phi_j(\boldsymbol{x}) - u_{\boldsymbol{a}_i}(\boldsymbol{x}_i) \right|^2. \tag{1.2}$$

By expanding a neural operator [29] output in the basis $\{\Phi_j\}_{j\in J}$, we can formally interpret its learning within the same framework. While expressive, such nonlinear architectures typically require many training samples and offer limited theoretical guarantees. In contrast, we adopt a simple linear feature model of the form $b_j(\boldsymbol{a}; \boldsymbol{c}) = \sum_{k\in K} c_{k,j}\Psi_k(\boldsymbol{a})$, where $\{\Psi_k\}_{k\in K}$ is a fixed dictionary of features $\Psi_k \colon \mathbb{R}^n \to \mathbb{R}$ and the coefficients $c_{k,j} \in \mathbb{R}$ are the learnable parameters. Setting $\boldsymbol{k} = (k, j) \in \Gamma \coloneqq K \times J$ and $T_{\boldsymbol{k}}(\boldsymbol{a}, \boldsymbol{x}) \coloneqq \Psi_k(\boldsymbol{a})\Phi_j(\boldsymbol{x})$, the resulting approximation (1.1) can be compactly written as $G_n[\boldsymbol{a}](\boldsymbol{x}) \approx \sum_{\boldsymbol{k}\in\Gamma} c_{\boldsymbol{k}} T_{\boldsymbol{k}}(\boldsymbol{a}, \boldsymbol{x})$. Then, the generic operator learning task (1.2) reduces to the least-squares problem

$$\operatorname*{arg\,min}_{\boldsymbol{c}\in\mathbb{R}^{|\Gamma|}} \sum_{i=1}^M \left| \sum_{\boldsymbol{k}\in\Gamma} c_{\boldsymbol{k}} T_{\boldsymbol{k}}(\boldsymbol{a}_i, \boldsymbol{x}_i) - u_{\boldsymbol{a}_i}(\boldsymbol{x}_i) \right|^2, \tag{1.3}$$

which is theoretically well analyzed. However, since the number of unknowns $|\Gamma|$ typically far exceeds the number of training samples M, solving (1.3) directly leads to severe overfitting. Sparse approximation methods [32, 11, 12, 25] address this by assuming that G_n can be represented accurately using only a small number of nonzero coefficients $c_{\boldsymbol{k}}$. This assumption is not merely a regularization device: the analysis of specific PDEs shows that the analytical solutions indeed satisfy such sparsity constraints [44]. Together, these observations motivate the constrained training problem

$$\operatorname*{arg\,min}_{I\subset\Gamma\colon |I|\le s} \; \min_{\boldsymbol{c}\in\mathbb{R}^{|I|}} \sum_{i=1}^M \left| \sum_{\boldsymbol{k}\in I} c_{\boldsymbol{k}} T_{\boldsymbol{k}}(\boldsymbol{a}_i, \boldsymbol{x}_i) - u_{\boldsymbol{a}_i}(\boldsymbol{x}_i) \right|^2. \tag{1.4}$$

The problem (1.4) can be solved, e.g., with greedy algorithms such as orthogonal matching pursuit (OMP) [43]. Under suitable conditions on the feature dictionary and the sample distribution, robust recovery guarantees for (1.4) are available; see, e.g., [53, 17, 39].

While sparse recovery addresses the overfitting problem, the cardinality of the search space Γ can itself be prohibitively large. For a product-type basis for Φ_j and Ψ_k with, e.g., 64 basis functions per direction, the number of candidate indices already grows to $|\Gamma| \approx 64^{n+d}$, rendering an exhaustive search computationally infeasible. As a remedy, the dimension-incremental framework of [45, 24, 23] constructs the support set I dimension by dimension: beginning with a one-dimensional candidate set, it successively extends the current support rather than searching Γ directly. The extension step involves solving several sparse recovery problems of the form (1.4), each with a tractable number of candidates. In this way, the method can efficiently exploit the underlying sparsity structure while considerably reducing both computational and sample complexity compared to an exhaustive search. A recent alternative for managing large (continuous) candidate sets is generative feature training [19], which adaptively resamples the candidates for (1.4) in place of a greedy approach.

Contributions Our contributions can be summarized as follows.

- We combine the dimension-incremental support detection framework of [45, 24] with OMP-based sparse recovery [39] to compute sparse approximations of PDE solution operators from carefully selected samples. Compared to cubature-based recovery [23], this significantly reduces the sample size and consequently the number of PDE solves.
- By design, the cubature-based recovery in [44] requires an independent PDE solve for each sample. In contrast, sparse recovery methods can naturally incorporate multiple samples with different spatial locations obtained from a single solve, which improves the computational efficiency even further.
- We numerically validate the proposed methods for several PDEs, comparing approximation quality, runtime, and sampling complexity against cubature-based sparse approximation methods and Tensorized Fourier Neural Operators (TFNOs) as a baseline.
- Unlike purely data-driven black-box approaches, our approach provides interpretable information about the parameter dependence of the solution via the detected index sets and their interaction structure.

Figure 1.1 provides a schematic overview of the proposed approximation strategies together with the methods used for comparison. Moreover, Table 1.1 summarizes typical parameter sizes occurring in the numerical experiments of Section 3, as well as the corresponding numbers of PDE solves required by the different approaches. The Python implementation and numerical experiment scripts are publicly available at [50].

Outline The remainder of this work is organized as follows. Section 2 discusses the parametric representation of PDE solution operators and their sparse approximation. Subsequently, we recall the dimension-incremental support detection framework and combine it with OMP-based coefficient recovery and an improved sampling strategy for PDE-based problems. Section 3 presents numerical experiments for the heat equation, Burgers' equation, and a parametric diffusion equation, comparing approximation quality, runtime, and sampling complexity against cubature-based approaches and TFNOs. Finally, Section 3.4 discusses the numerical results and the trade-offs between approximation quality and computational cost.

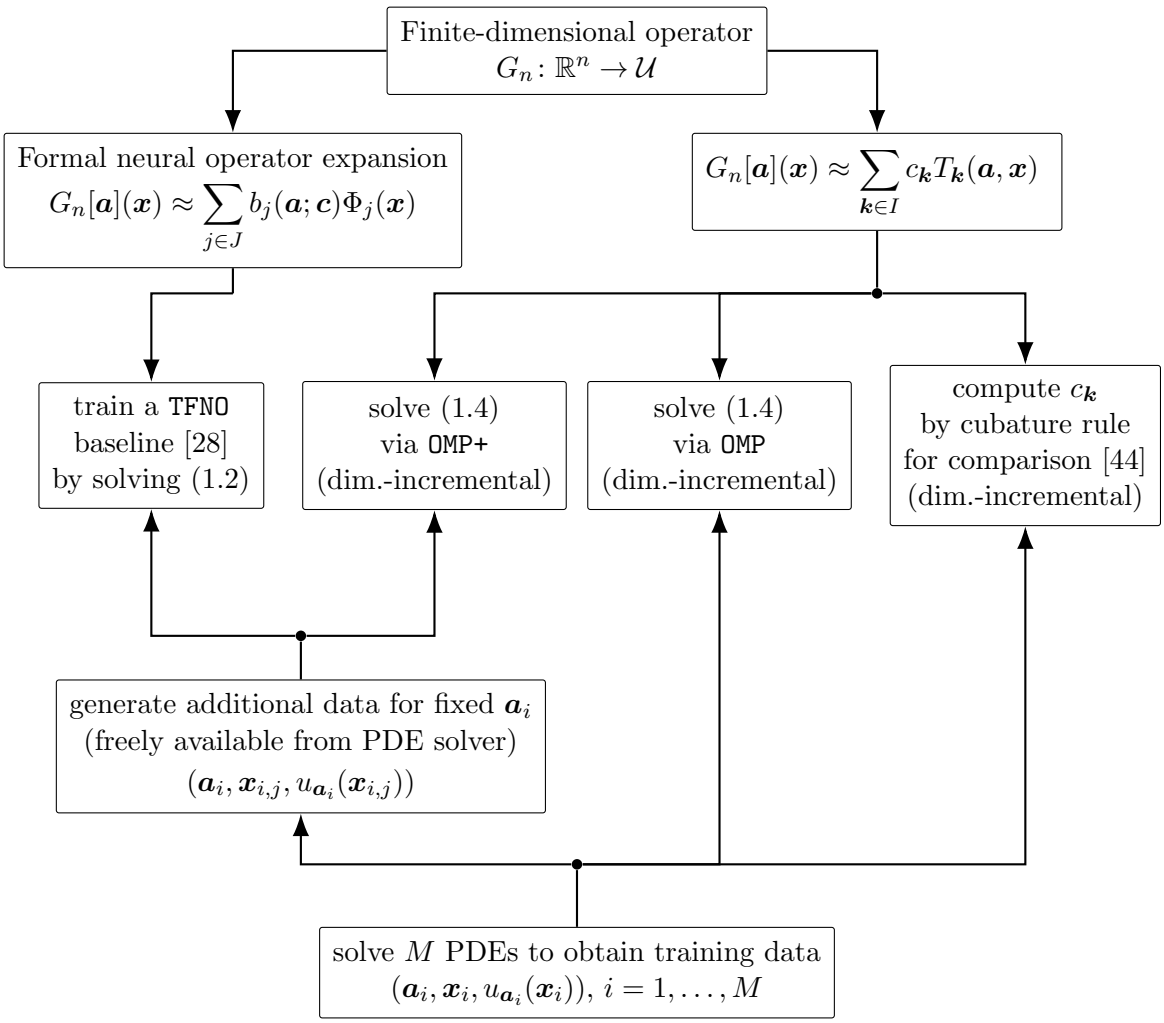


Figure 1.1: Schematic comparison of the considered approximation strategies.

quantity	Heat eq.	Burgers' eq.	Parametric diffusion eq.
overall dimension d_{ex}	11	10	22
amount of candidates $\lvert\Gamma\rvert$	$8.75 \cdot 10^{19}$	$1.35 \cdot 10^{18}$	$7.55 \cdot 10^{17*}$
target sparsity s	1000	1000	1000
TFNO parameters	529 233	81 049	529 233
PDE solves			
OMP	285 000	255 000	615 000
OMP+	35 006	35 002	79 006
cMR1L	702 054	5 706 215†	3 683 946
TFNO	40 500	40 500	86 000

* Additional assumption: $\|\boldsymbol{k}\|_0 \leq 7 \quad \forall \boldsymbol{k} \in \Gamma$.
† Here, $s = 500$ and a smaller Γ were used due to computational restrictions.

Table 1.1: Parameter counts and number of PDE solves for our experiments in Section 3.

Related work The approximation of (infinite-dimensional) operators from training data has recently attracted significant attention, with approaches broadly falling into three categories: linear expansions, neural operators, and (random) kernel methods.

From the viewpoint of approximation theory, operator learning can be cast as finding the coefficients of a basis expansion [13, 12]. This perspective is closely related to sparse polynomial approximation [11, 46]: if the solution operator admits a sparse representation in a suitable basis, only a small number of coefficients need to be recovered [2]. To this end, a PDE-oriented recovery strategy was investigated in [44]. Alternatively, one can directly learn mappings between the basis expansion coefficients of input and output functions using neural networks [26, 20] or approximate the Green's function instead [9].

Neural operators are typically trained in a supervised setting using (potentially physics-informed [36]) loss functions. Prominent architectures include DeepONets [38, 52] and Fourier Neural Operators (FNOs) [34], many of which are implemented as part of the NeuralOperator library [27]. Spectral neural operators and related architectures replace purely Fourier-based representations with polynomial or spectral basis expansions [16, 37], a perspective closely related to this work. Other recent directions include the incorporation of attention mechanisms [18, 35] and the aim for foundation models [49]. Comprehensive surveys covering practical and theoretical aspects of neural operators can be found in [10, 31, 14].

Operator-valued kernels provide a principled framework [21], though the direct implementation is computationally prohibitive in general. Random feature approaches [42] offer a scalable alternative: by fixing the random parameters during training, the optimization reduces to a linear problem. Interestingly, kernel-based methods can achieve accuracy competitive with neural operators [6]. A related Gaussian process framework was recently proposed in [40].

Finally, we remark that there is also growing interest in topics such as error estimates [33, 29], discretization consistency [5] and sample complexity [30] of neural operators.

2 Theory

The presentation of the general framework follows [44], where operator learning for partial differential equations (PDEs) is investigated from a sparse high-dimensional approximation perspective. For the purposes of this paper, we adapt their approach and incorporate dedicated sparse recovery algorithms. While we restrict ourselves to time-dependent PDEs as a running example below, we stress that the proposed method applies to a broad class of operators, which also includes stationary and parametric PDEs.

2.1 Time-dependent partial differential equations

Let $\Omega \subset \mathbb{R}^d$ be a bounded spatial domain and $T > 0$. Further, assume that we are given function spaces $\mathcal{U}$ and $\mathcal{F}$ defined on the space-time domain $\Omega \times (0,T)$, a function space $\mathcal{H}$ on $\partial\Omega \times (0,T)$, and a function space $\mathcal{U}_0$ on Ω. Here, the pairs $(\mathcal{U},\mathcal{U}_0)$ and $(\mathcal{U},\mathcal{H})$ should be compatible in the sense that $u(\cdot,0) \in \mathcal{U}_0$ and $u|_{\partial\Omega\times(0,T)} \in \mathcal{H}$ for any $u \in \mathcal{U}$. Within this setting, we consider differential equations of the form

$$Lu = f \qquad \text{s.t. } u|_{\partial\Omega\times(0,T)} = h \text{ and } u(\cdot,0) = u_0, \tag{2.1}$$

where $L\colon \mathcal{U} \to \mathcal{F}$ is a (possibly nonlinear) differential operator, $f \in \mathcal{F}$ is an inhomogeneity, and $u_0 \in \mathcal{U}_0$ is an initial value. Throughout, we assume that (2.1) is well-posed in the sense

that for every $(f, h, u_0) \in \mathcal{F} \times \mathcal{H} \times \mathcal{U}_0$, there exists a unique solution $u \in \mathcal{U}$ depending continuously on the data (f, h, u_0). Then, (2.1) induces a solution operator

$$G\colon \mathcal{F} \times \mathcal{H} \times \mathcal{U}_0 \to \mathcal{U}, \qquad (f, h, u_0) \mapsto u. \tag{2.2}$$

In practice, often only a subset of the data (f, h, u_0) is varied, while the remaining components are fixed. In particular, a common situation is that the solution operator G reduces to a mapping from the initial data u_0 to the solution u, with f and h being fixed. Then, our goal is to approximate G by a surrogate model that enables fast evaluations for new input data u_0 without the need for solving the PDE (2.1).

Remark 2.1. *For the Poisson equation as discussed in [44, Sec. 3], the solution would depend only on the spatial variable $\boldsymbol{x}$ and the inhomogeneity f. Such variations mainly influence the dimensionality of the product spaces in* (2.2) *and the choice of discretization, while leaving the approximation procedure for G described below unchanged.*

2.2 Parametric operator representation and sparse approximation

In order to obtain a finite-dimensional representation of the domain of G, we expand the initial condition u_0 as

$$u_0(\boldsymbol{x}) \approx \sum_{j=1}^{n} a_j A_j(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega, \tag{2.3}$$

with basis functions $A_j\colon \Omega \to \mathbb{R}$, $j = 1, \dots, n$, and coefficients $\boldsymbol{a} = (a_1, \dots, a_n) \in [-1, 1]^n$. Note that the chosen functions A_j should lead to a sufficiently accurate approximation in $\mathcal{U}_0$. Based on (2.3), we get the induced solution operator $G_n\colon [-1, 1]^n \to \mathcal{U}$ with $\boldsymbol{a} \mapsto u$.

In general, any appropriate discretization G_n of G can be identified with a function defined on an extended bounded product domain $\mathcal{D} \coloneqq \prod_{j=1}^{d_{\mathrm{ex}}} \mathcal{D}_j \subset \mathbb{R}^{d_{\mathrm{ex}}}$. In the case of (2.3), we get $\mathcal{D} = \Omega \times [0, T] \times [-1, 1]^n$. For the next step, we assume that the induced function admits an expansion in a bounded orthonormal product basis (BOPB) $\{\Phi_{\boldsymbol{k}} : \boldsymbol{k} \in \mathbb{N}^{d_{\mathrm{ex}}}\} \subset L_2(\mathcal{D}, \mu)$, where μ is a product measure. Recall that a BOPB consists of tensor-products $\Phi_{\boldsymbol{k}}(\boldsymbol{y}) = \prod_{j=1}^{d_{\mathrm{ex}}} \phi_{j,k_j}(y_j)$. More details can be found, e.g., in [23].

For the operator G_n induced by (2.3), the BOPB expansion reads as

$$u(\boldsymbol{x}, \tau; \boldsymbol{a}) = \sum_{\boldsymbol{k} \in \mathbb{N}^{d_{\mathrm{ex}}}} c_{\boldsymbol{k}}\, \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \tau, \boldsymbol{a}), \qquad (\boldsymbol{x}, \tau, \boldsymbol{a}) \in \mathcal{D}, \tag{2.4}$$

with $c_{\boldsymbol{k}} \in \mathbb{C}$ being the corresponding basis coefficients

$$\hat{c}_{\boldsymbol{k}} = \int_{\mathcal{D}} g(\boldsymbol{x}, \tau, \boldsymbol{a}) \overline{\Phi_{\boldsymbol{k}}(\boldsymbol{x}, \tau, \boldsymbol{a})} \mathrm{d}(\boldsymbol{x}, \tau, \boldsymbol{a}). \tag{2.5}$$

Then, our aim is to accurately approximate u (if possible) by a sparse truncation

$$S_I u(\boldsymbol{x}, \tau; \boldsymbol{a}) = \sum_{\boldsymbol{k} \in I} c_{\boldsymbol{k}}\, \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \tau, \boldsymbol{a}),$$

where $I \subset \mathbb{N}_0^{d_{\mathrm{ex}}}$ is a finite index set. Since the exact $c_{\boldsymbol{k}}$ are unknown, we replace them by approximations $\hat{u}_{\boldsymbol{k}}$ (for which we outline a computation procedure below) and obtain

$$S_I^{\mathcal{A}} u(\boldsymbol{x}, \tau, \boldsymbol{a}) = \sum_{\boldsymbol{k} \in I} \hat{u}_{\boldsymbol{k}}\, \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \tau, \boldsymbol{a}). \tag{2.6}$$

In summary, (2.6) yields a continuous approximation of G by first projecting u_0 onto its finite coefficient vector $\boldsymbol{a}$ and then evaluating $S_I^{\mathcal{A}} u(\cdot, \cdot, \boldsymbol{a})$. The resulting approximation error can be split into a truncation part and a coefficient recovery part. More precisely, using the uniform boundedness of the basis functions $\Phi_{\boldsymbol{k}}$, we obtain

$$\begin{aligned}
\left\| u - S_I^{\mathcal{A}} u \right\|_\infty &\leq \left\| u - S_I u \right\|_\infty + \left\| S_I u - S_I^{\mathcal{A}} u \right\|_\infty \\
&\leq \sum_{\boldsymbol{k} \notin I} |c_{\boldsymbol{k}}| \left\| \Phi_{\boldsymbol{k}} \right\|_\infty + \sum_{\boldsymbol{k} \in I} |c_{\boldsymbol{k}} - \hat{u}_{\boldsymbol{k}}| \left\| \Phi_{\boldsymbol{k}} \right\|_\infty \\
&\leq \sup_{\boldsymbol{k} \in I} \left\| \Phi_{\boldsymbol{k}} \right\|_\infty \left(\sum_{\boldsymbol{k} \notin I} |c_{\boldsymbol{k}}| + \sum_{\boldsymbol{k} \in I} |c_{\boldsymbol{k}} - \hat{u}_{\boldsymbol{k}}| \right).
\end{aligned}$$

Hence, a good approximation requires both identifying a suitable index set I and an accurate method for estimating the coefficients $c_{\boldsymbol{k}}$ from samples $u(\boldsymbol{x}_j, \tau_j, \boldsymbol{a}_j)$. For our running example, we keep in mind that samples are obtained by numerically solving the PDE (2.1) for the parameters $\boldsymbol{a}_j$, which makes sampling costly.

Given a finite index set I, the corresponding coefficients $c_{\boldsymbol{k}}$, $\boldsymbol{k} \in I$, can be estimated using a cubature formula based on (2.5). Alternatively, they may be obtained by solving the linear system that arises from inserting the samples $u(\boldsymbol{x}_j, \tau_j, \boldsymbol{a}_j)$ into (2.4), for instance via a standard least squares approach. However, since the index set I is not known a priori, one typically has to work with the product space $\Gamma \supset I$ instead. Without imposing strong a priori assumptions, such search spaces grow rapidly in high-dimensional settings and render the aforementioned approaches computationally infeasible. We therefore turn to the problem of determining a suitable index set $I \subset \Gamma \subset \mathbb{N}_0^{d_{\text{ex}}}$.

Remark 2.2. *In our implementation, we transform the problem-specific product domain $\mathcal{D}$ to the hypercube $[-1, 1]^{d_{\text{ex}}}$ via problem-dependent affine maps, specified for each PDE individually. After this transformation, Chebyshev polynomials can serve as a BOPB for the function induced by the operator G_n.*

2.3 Dimension-incremental support detection and sparse recovery

Since the candidate set Γ has product structure, its cardinality scales as $10^{d_{\text{ex}}}$ even with only 10 basis functions per dimension. For $d_{\text{ex}} = 12$, this already yields 10^{12} candidate index tuples, making an exhaustive search for a suitable I in general computationally infeasible.

As an alternative, we employ the dimension-incremental support detection strategy proposed in [23]. The main idea is to build I progressively, starting from one-dimensional projections and then combining the detected candidates across dimensions. Below we provide a brief summary; see Algorithm 1 and [23] for further details and a detection guarantee for a suitable index set I. To simplify the notation, we switch from the problem-specific function u to a generic function g, and correspondingly collect all variables into a single d_{ex}-dimensional input vector.

Let $g \colon \mathbb{R}^{d_{\text{ex}}} \to \mathbb{C}$ be an (unknown) target function with (s-sparse) expansion

$$g = \sum_{\boldsymbol{k} \in I} \hat{g}_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}, \tag{2.7}$$

where I with $|I| = s$ is contained in a prescribed finite candidate space $\Gamma \subset \mathbb{N}_0^{d_{\text{ex}}}$. Typically, Γ is simply a cube in $\mathbb{N}_0^{d_{\text{ex}}}$. For each coordinate $t \in \{1, \ldots, d_{\text{ex}}\}$, we consider the one-dimensional

projected candidate set $\mathcal{P}_{\{t\}}(\Gamma) \coloneqq \{l \in \mathbb{N}_0 \,|\, \exists \boldsymbol{k} \in \Gamma : k_t = l\}$ and for each $l \in \mathcal{P}_{\{t\}}(\Gamma)$ the so-called projected coefficients

$$\hat{g}_{\{t\},l}(\boldsymbol{\theta}) \coloneqq \int_{\mathcal{D}_t} g_{\{t\}}(\xi, \boldsymbol{\theta}) \overline{\phi_{t,l}(\xi)} \,\mathrm{d}\xi = \sum_{\boldsymbol{k} \in I : k_t = l} \hat{g}_{\boldsymbol{k}} \prod_{j \neq t} \phi_{j,k_j}(\theta_j), \tag{2.8}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{d_{\mathrm{ex}}-1}$ is an anchor and $g_{\{t\}}(\xi, \boldsymbol{\theta})$ denotes the value of g with ξ in the t-th dimension and $\boldsymbol{\theta}$ for the remaining coordinates. Due to the representation (2.7), at most s coefficients $\hat{g}_{\{t\},l}(\boldsymbol{\theta})$ should be nonzero. Thus, among all coefficients $l \in \mathcal{P}_{\{t\}}(\Gamma)$, we retain the s largest ones (in terms of $|\hat{g}_{\{t\},l}(\boldsymbol{\theta})|$) exceeding a prescribed threshold, and discard all others as negligible. All detected candidates l are collected in the temporary index sets $I_{\{t\}}$.

Remark 2.3. *The projected coefficients $\hat{g}_{\{t\},l}(\boldsymbol{\theta})$ are an indicator for the relevance of the index l. However, an unfavorable choice of the anchor $\boldsymbol{\theta}$ may cause cancellations in* (2.8)*, leaving some relevant indices undetected. A remedy is to perform $r \in \mathbb{N}$ detection iterations with distinct anchors $\boldsymbol{\theta}$ and to consider the union of the detected indices. While lower bounds on r required to achieve a prescribed failure probability can be derived analogously to [23, Thm. 3.5], in practice, considerably smaller values of r often suffice for reliable results.*

Unfortunately, increasing r raises both the required sample size (and thus the number of PDE solves) and the overall computational effort. To mitigate this, the anchors $\boldsymbol{\theta} \in \mathbb{R}^{d_{\mathrm{ex}}-1}$ can be selected using a distance-maximizing strategy rather than purely random sampling. This minimizes the risk of multiple anchors capturing redundant local behavior of $\hat{g}_{\{t\},l}(\boldsymbol{\theta})$. Such local dependencies occur in the heat equation example in Section 3.1, where certain solution components decay rapidly over time, rendering anchors with large t blind to them.

Next, for $t \geq 2$, the candidate sets are combined incrementally. Given $I_{\{1,\dots,t-1\}}$, one forms the sets

$$\begin{aligned} \mathcal{P}_{\{1,\dots,t\}}(\Gamma) &\coloneqq \{\boldsymbol{l} \in \mathbb{N}_0^t \,|\, \exists \boldsymbol{k} \in \Gamma \subset \mathbb{N}_0^{d_{\mathrm{ex}}} : \boldsymbol{k}_{\{1,\dots,t\}} = \boldsymbol{l}\} \\ K &\coloneqq \big(I_{\{1,\dots,t-1\}} \times I_{\{t\}}\big) \cap \mathcal{P}_{\{1,\dots,t\}}(\Gamma), \end{aligned} \tag{2.9}$$

and investigates the corresponding projected coefficients on K as in the one-dimensional case. In particular, we again consider the projected coefficients

$$\hat{g}_{\{1,\dots,t\},\boldsymbol{l}}(\boldsymbol{\theta}) \coloneqq \int_{\prod_{j=1}^t \mathcal{D}_j} g_{\{1,\dots,t\}}(\boldsymbol{\xi}, \boldsymbol{\theta}) \overline{\Phi_{\{1,\dots,t\},\boldsymbol{l}}(\boldsymbol{\xi})} \,\mathrm{d}\boldsymbol{\xi} = \sum_{\boldsymbol{k} \in I : \boldsymbol{k}_{\{1,\dots,t\}} = \boldsymbol{l}} \hat{g}_{\boldsymbol{k}} \prod_{j \notin \{1,\dots,t\}} \Phi_{j,k_j}(\theta_j), \tag{2.10}$$

estimate their absolute value, retain the s largest ones exceeding the threshold, and collect them in the index set $I_{\{1,\dots,t\}}$. In this way, higher-order interactions are identified successively until the final index set $I_{\mathrm{est}} \coloneqq I_{\{1,\dots,d_{\mathrm{ex}}\}}$ is obtained.

In [44], the projected coefficients defined in (2.8) and (2.10) are approximated via cubature rules, which typically require a large number of sampling points $\boldsymbol{\xi}^{(j)}$ at carefully chosen locations. Specifically, so-called rank-1 lattices are employed, which provide accurate (and in some cases even exact) cubature for functions with suitable smoothness or frequency structure. When using large lattices, this enables reliable and efficient recovery of the projected coefficients for all candidate indices, including the ones that are actually negligible.

Algorithm 1 Dimension-Incremental Index Detection (Simplified)

Input: Search space $\Gamma \subset \mathbb{N}^{d_{\text{ex}}}$, target function g, sparsity level $s \in \mathbb{N}$, detection threshold $\delta_+ > 0$, number of detection iterations $r \in \mathbb{N}$.

(Step 1) Single component identification

for $t = 1, \dots, J$ **do**
  Set $I_{\{t\}} \coloneqq \emptyset$.
  Draw random sampling nodes $(\xi^{(j)})_{j=1}^{M} \subset \mathcal{D}_t$.
  for $i = 1, \dots, r$ **do**
    Draw a random anchor $\boldsymbol{\theta} \in \prod_{j=1, j\neq t}^{d_{\text{ex}}} \mathcal{D}_j$.
    Sample function values $g_{\{t\}}(\xi^{(j)}, \boldsymbol{\theta})$.
    Perform sparse reconstruction of $\hat{g}_{\{t\},l}(\boldsymbol{\theta})$ for all $l \in \mathcal{P}_{\{t\}}(\Gamma)$.
    Add the s largest indices l satisfying $|\hat{g}_{\{t\},l}(\boldsymbol{\theta})| \geq \delta_+$ to $I_{\{t\}}$.
  end for
end for

(Step 2) Coupled component identification

for $t = 2, \dots, J$ **do**
  Set $\tilde{r} \coloneqq r$ if $t < J$ and $\tilde{r} \coloneqq 1$ otherwise.
  Set $K \coloneqq (I_{\{1,\dots,t-1\}} \times I_{\{t\}}) \cap \mathcal{P}_{\{1,\dots,t\}}(\Gamma)$.
  Draw random sampling nodes $(\boldsymbol{\xi}^{(j)})_{j=1}^{M} \subset \prod_{j=1}^{t} \mathcal{D}_j$.
  Set $I_{\{1,\dots,t\}} \coloneqq \emptyset$.
  for $i = 1, \dots, \tilde{r}$ **do**
    Draw a random anchor $\boldsymbol{\theta} \in \prod_{j=t+1}^{d_{\text{ex}}} \mathcal{D}_j$.
    Sample function values $g_{\{1,\dots,t\}}(\boldsymbol{\xi}^{(j)}, \boldsymbol{\theta})$ (or $g_{\{1,\dots,t\}}(\boldsymbol{\xi}^{(j)})$ if $t = J$).
    Perform sparse reconstruction of $\hat{g}_{\{1,\dots,t\},\boldsymbol{l}}(\boldsymbol{\theta})$ for all $\boldsymbol{l} \in K$.
    Add the s largest indices $\boldsymbol{l}$ where $|\hat{g}_{\{1,\dots,t\},\boldsymbol{k}}(\boldsymbol{\theta})| \geq \delta_+$ to $I_{\{1,\dots,t\}}$.
  end for
end for

(Step 3)

Set $I_{\text{est}} \coloneqq I_{\{1,\dots,J\}}$ and $\hat{g}_{\boldsymbol{l}} \coloneqq \hat{g}_{\{1,\dots,J\},\boldsymbol{l}}$ for all $\boldsymbol{l} \in I$.

Output: Detected index set $I_{\text{est}} \subset \Gamma$, approximated coefficients $(\hat{g}_{\boldsymbol{l}})_{\boldsymbol{l} \in I} \in \mathbb{C}^{|I|}$.

In contrast to this exhaustive approximation, we propose to directly exploit the system's inherent sparsity. More precisely, (2.7) implies that also the projected expansions

$$g_{\{t\}}(\xi, \boldsymbol{\theta}) = \sum_{l \in \mathcal{P}_{\{t\}}(\Gamma)} \hat{g}_{\{t\},l}(\boldsymbol{\theta}) \Phi_{\{t\},l}(\xi), \tag{2.11}$$

and similarly the ones with (2.10), are s sparse. Classical sparse recovery algorithms, such as the orthogonal matching pursuit (OMP) [32] discussed in Section 2.4, typically require only around $M = 5s$ random sample locations $\boldsymbol{\xi}^{(j)}$, thereby considerably improving the computational efficiency of Algorithm 1 compared to [44]. Since our approach treats g as a black box, any numerical scheme, such as finite elements, finite differences, or spectral methods, can be used to compute the function values $g_{\{1,\dots,t\}}(\boldsymbol{\xi}^{(j)}, \boldsymbol{\theta})$. In turn, the accuracy of these evaluations directly affects the quality of the computed approximation (2.6). Finally, we emphasize that the sampled PDE input parameters $\boldsymbol{a}_i$ must conform to the anchor decomposition: each $\boldsymbol{a}_i$ is partitioned into active coordinates, sampled randomly, and fixed anchor coordinates.

Algorithm 2 OMP

Input: Sampled BOPB matrix $\boldsymbol{\Phi}$, sample vector $g_{\{1,\dots,t\}}(\boldsymbol{\xi}) \in \mathbb{R}^M$, number of iterations s

1: $S^{(0)} = \emptyset$
2: **for all** $k = 0, \dots, s-1$ **do**
3: $\quad \boldsymbol{l}^{(k+1)} = \operatorname{argmax}_{\boldsymbol{l} \in K} \{|(\boldsymbol{\Phi}^*(\boldsymbol{\Phi}\boldsymbol{z}^{(k)} - g_{\{1,\dots,t\}}(\boldsymbol{\xi}))_{\boldsymbol{l}}|\}$
4: $\quad S^{(k+1)} = S^{(k)} \cup \{\boldsymbol{l}^{(k+1)}\}$
5: $\quad \boldsymbol{z}^{(k+1)} = \operatorname{argmin}_{\boldsymbol{z} \in \mathbb{C}^{|J|}} \{\|\boldsymbol{\Phi}\boldsymbol{z} - g_{\{1,\dots,t\}}(\boldsymbol{\xi})\|_{\ell_2} \quad \text{s.t. } \operatorname{supp}(\boldsymbol{z}) \subset S^{(k+1)}\}$
6: **end for**
7: **return** $\boldsymbol{z}^{(s)}$

2.4 Orthogonal Matching Pursuit

Now, we briefly discuss OMP, which is used to reconstruct the relevant projected coefficients in lines 7 and 19 of Algorithm 1. This algorithm is a greedy approach developed for sparse approximation and compressed sensing. For a broad overview, we refer to [43, 51, 32]. Assume, we are in the t-th dimension-incremental step of Algorithm 1 and consider the candidate set K as given in (2.9) and sampling values $g_{\{1,\dots,t\}}(\boldsymbol{\xi}) \coloneqq (g_{\{1,\dots,t\}}(\boldsymbol{\xi}^{(j)}))_{j=1}^M \in \mathbb{R}^M$. Starting from an empty active set, OMP iteratively selects at each step the product of basis functions $\phi_{\{1,\dots,t\},\boldsymbol{l}}$ for which the corresponding column of the basis matrix

$$\boldsymbol{\Phi} \coloneqq (\phi_{\{1,\dots,t\},\boldsymbol{l}}(\boldsymbol{\xi}^{(j)})_{j=1,\dots,M,\, \boldsymbol{l} \in K} \in \mathbb{C}^{M \times |K|} \tag{2.12}$$

is most strongly correlated with the current residual. The active set is then updated with the index $\boldsymbol{l}$, and the basis coefficients $\hat{g}_{\{1,\dots,t\},\boldsymbol{l}}(\boldsymbol{\theta})$ restricted to this set are determined via a least squares solve. This procedure is repeated until the prescribed sparsity s is reached or a suitable stopping criterion, e.g., an absolute or relative error tolerance, is satisfied.

Under suitable assumptions on the sampling points $(\boldsymbol{\xi}^{(j)})_{j=1}^M$, the resulting Algorithm 2 is guaranteed to recover the correct support of the basis coefficients $\hat{g}_{\{1,\dots,t\},\boldsymbol{l}}(\boldsymbol{\theta})$, $\boldsymbol{l} \in K$, with high probability provided that the number of samples scales like $M \gtrsim s \log |K|$. The complexity of a single iteration can be bounded by $\mathcal{O}(M|K| + Ms + s^2)$, cf. [47], which is dominated by the cost of the matrix-vector multiplication $(M|K|)$. Performing s iterations leads to the overall computational complexity $\mathcal{O}(sM|K|)$. Our implementation is based on the methodology presented in [39], with a Python implementation being publicly available[1].

Remark 2.4. *While we only discuss OMP here, other sparse recovery methods can be also employed. Examples include the CoSaMP [41] and square-root LASSO [7]. Our Python implementation [50] includes these approaches as additional choices. In our numerical experiments, no method consistently outperformed the others such that we have chosen OMP for simplicity.*

2.5 Improving Sample Efficiency (OMP+)

In Algorithm 1, each evaluation of u at a node $(\boldsymbol{x}^{(j)}, \tau^{(j)}, \boldsymbol{a}^{(j)}) \in \mathcal{D}$ requires a separate numerical solve of (2.1) if the parameter instances $\boldsymbol{a}^{(j)}$ differ. As described so far, Algorithm 1 incorporates random samples $\{\boldsymbol{\xi}^{(j)}\}_{j=1}^M$ conditional to a random anchor, and thus in principle only a single value $u(\boldsymbol{x}^{(j)}, \tau^{(j)})$ from each PDE solve, even though the solution is typically

[1] `https://github.com/Zeppo1994/SparseRecovery`

available at many grid points $(\boldsymbol{x}, \tau)$. Given the substantial cost of PDE solves, this represents a significant inefficiency. Fortunately, in contrast to the cubature-based approach in [44], OMP imposes no structural constraints on the sampling nodes $\{\boldsymbol{\xi}^{(j)}\}_{j=1}^{M} \in \mathbb{R}^t$. Thus, we can systematically incorporate multiple samples from a computed solution, as detailed below.

Consider a t-th dimension-incremental step with $t > d+1$ in Algorithm 1. Then, we have $\boldsymbol{\xi}^{(j)} = (\boldsymbol{x}^{(j)}, \tau^{(j)}, \boldsymbol{a}_1^{(j)})$ with $\boldsymbol{a}_1^{(j)} \in \mathbb{R}^{t-d-1}$ and the anchor $\boldsymbol{\theta} = \boldsymbol{a}_2 \in \mathbb{R}^{n+d+1-t}$. Instead of evaluating the PDE solution $u(\cdot,\cdot,\boldsymbol{a}^{(j)})$ for $\boldsymbol{a}^{(j)} = (\boldsymbol{a}_1^{(j)}, \boldsymbol{a}_2)$, $j = 1, \dots, M$, only at the sampled $(\boldsymbol{x}^{(j)}, \tau^{(j)})$, we additionally evaluate them at b^{d+1} randomly chosen points $(\boldsymbol{x}, \tau) \in \mathbb{R}^{d+1}$, which of course differ for each $\boldsymbol{a}^{(j)}$. We recommend $b = 3$ as default. The number of evaluations per solve must be carefully chosen to balance the exploration of the parameter and space-time domains. A smaller amount of space-time evaluations requires a higher number of costly PDE solves to achieve a given target sample size. Conversely, too many evaluations per parameter result in insufficient coverage of the parameter domain (the extreme case being only a single parameter). Moreover, since the complexity of OMP scales linearly with M, adding more space-time evaluations is only recommended when they contain non-redundant information. In our experiments, we refer to this modified version of `OMP` as `OMP+`.

Remark 2.5. *In all one-dimensional steps of Algorithm 1, where a coordinate of $(\boldsymbol{x}, \tau)$ is the active variable, as well as in all dimension-incremental steps with $t \leq d+1$, the random anchor $\boldsymbol{\theta}$ fully covers the parameters $\boldsymbol{a}$. In these cases, a single PDE solve suffices to generate the samples. While this applies to any reconstruction method, we only incorporate it into `OMP+`. The latter already distinguishes between spatial, temporal, and parameter coordinates, making the modification straightforward to implement. In contrast, `OMP` follows the original non-intrusive dimension-incremental framework, which does not have access to structural information.*

3 Numerics

We benchmark Algorithm 1 for several PDEs, in both its standard version `OMP` and the modified variant `OMP+` from Section 2.5. For $\boldsymbol{a} \in [-1,1]^n$, we measure the approximation accuracy in terms of the absolute ℓ_2-error

$$\mathrm{err}_{\mathrm{abs}}(\boldsymbol{a}) \coloneqq \left\| S^{\mathcal{A}}_{I_{\mathrm{est}}} u(\boldsymbol{x},\tau,\boldsymbol{a}) - u(\boldsymbol{x},\tau,\boldsymbol{a}) \right\|_{\ell_2} = \left(\sum_{j=1}^{G} \left| S^{\mathcal{A}}_{I_{\mathrm{est}}} u(\boldsymbol{x}^{(j)},\tau^{(j)},\boldsymbol{a}) - u(\boldsymbol{x}^{(j)},\tau^{(j)},\boldsymbol{a}) \right|^2 \right)^{\frac{1}{2}}, \tag{3.1}$$

which coincides with the quality criterion in Algorithm 1, and the (scale-invariant) relative ℓ_2-error

$$\begin{aligned} \mathrm{err}_{\mathrm{rel}}(\boldsymbol{a}) &\coloneqq \frac{\left\| S^{\mathcal{A}}_{I_{\mathrm{est}}} u(\boldsymbol{x},\tau,\boldsymbol{a}) - u(\boldsymbol{x},\tau,\boldsymbol{a}) \right\|_{\ell_2}}{\left\| u(\boldsymbol{x},\tau,\boldsymbol{a}) \right\|_{\ell_2}} \\ &= \frac{\left(\sum_{j=1}^{G} \left| S^{\mathcal{A}}_{I_{\mathrm{est}}} u(\boldsymbol{x}^{(j)},\tau^{(j)},\boldsymbol{a}) - u(\boldsymbol{x}^{(j)},\tau^{(j)},\boldsymbol{a}) \right|^2 \right)^{\frac{1}{2}}}{\left(\sum_{j=1}^{G} \left| u(\boldsymbol{x}^{(j)},\tau^{(j)},\boldsymbol{a}) \right|^2 \right)^{\frac{1}{2}}}. \end{aligned} \tag{3.2}$$

In (3.1) and (3.2), the $(\boldsymbol{x}^{(j)}, \tau^{(j)})$, $j = 1, \dots, G$, are equidistant grid points in the space-time domain $\Omega \times [0,T]$ (or spatial domain Ω for stationary problems). We statistically compare

these quantities for several randomly drawn coefficients $\boldsymbol{a}$ in terms of range (min and max), median, first quartile, and third quartile.

Throughout, we employ the tensorized Chebyshev polynomial basis, which is given by

$$T_{\boldsymbol{k}}(\boldsymbol{z}) \coloneqq \prod_{j=1}^{d_{\mathrm{ex}}} T_{k_j}(z_j) \quad \text{with} \quad T_{k_j}(z_j) \coloneqq \begin{cases} 1 & k_j = 0 \\ \sqrt{2}\cos(k_j \arccos(z_j)) & k_j \neq 0 \end{cases},$$

where $d_{\mathrm{ex}} = d + 1 + n$ denotes the dimension of the space-time-parameter domain as in Section 2.2. This choice is motivated by the fact that several classes of parameter-dependent PDEs admit sparse or approximately sparse representations in terms of Chebyshev polynomials [44]. If not stated otherwise, Algorithm 1 uses the following parameters:

- the search space Γ is the (non-negative) full grid $[0, N]^{d+1+n}$ in $d + 1 + n$ dimensions (or $d + n$ for stationary examples) with extension N;
- the detection threshold $\delta_+ = 10^{-12}$;
- $r = 5$ detection iterations for `OMP` and $r = 2$ for `OMP+`, see also Remark 2.3;
- an individual sparsity level s for each experiment.

See [23] for more detailed information on these parameters and how they affect the behavior of the support detection. For the one-dimensional detection steps, we use s samples. This suffices because typically $N \ll s$, which ensures that the number of candidates in these steps (namely $N + 1$) remains much smaller than s. For the coupled detection steps, the standard `OMP` approach utilizes an oversampling factor of $c = 5$, leading to $5s$ samples and PDE solves. In contrast, the `OMP+` approach utilizes 3^{d+1} (or 3^d for stationary problems) additional space-time evaluations as discussed in Section 2.5, reducing the requirement to only s PDE solves. In both cases, the resulting sparse recovery problems (2.11) are solved using the OMP Algorithm 2 detailed in Section 2.4.

We compare our methods against the dimension-incremental method using Chebyshev multiple rank-1 lattices (`cMR1L`) [44, Sec. 3] that inspired our work. Further, we consider Tensorized Fourier Neural Operators (`TFNO`), implemented in PyTorch using the NeuralOperator library [27]. Given training samples $(u_0^{(j)}, u^{(j)})$, the underlying Fourier Neural Operator (FNO) framework learns the solution operator $G\colon u_0 \mapsto u$ from data by minimizing the supervised Sobolev loss

$$\min_{\theta} \frac{1}{M} \sum_{j=1}^{M} \|G_\theta(u_0^{(j)}) - u^{(j)}\|_{H^1}^2.$$

The neural operator G_θ is realized with a pointwise lifting layer, followed by a composition of multiple mappings (frequently called Fourier layers) of the form

$$v_{k+1}(\boldsymbol{x}) = \sigma\left(W_k v_k(\boldsymbol{x}) + \mathcal{F}^{-1}\big(R_k \cdot \mathcal{F}(v_k)\big)(\boldsymbol{x})\right),$$

where R_k are learned Fourier multipliers on a truncated set of modes and W_k are pointwise linear mappings, and a final pointwise projection layer. For more details on FNOs and their tensorized variants, we refer to [34, 28]. All `TFNO` models use 4 Fourier layers with 64 hidden channels and are trained using the AdamW optimizer with initial learning rate 10^{-3} and

weight decay 10^{-4}. Training is performed for 1000 epochs with a batch size of 64 using random parameter samples $\boldsymbol{a}_i$ with corresponding solutions $u_{\boldsymbol{a}_i}$ evaluated at resolution 100. Further, we incorporate a cosine annealing schedule that decays the learning rate to 10^{-8} and is stepped after each epoch.

All experiments are performed in Python$^{\text{TM}}$ and can be found together with the algorithm in [50]. Unless stated otherwise, all computations are carried out on two AMD EPYC 9534 64-Core processors. To accelerate the sampling process, parallelization with up to 96 workers is employed. The `TFNO` models are trained separately on a NVIDIA GeForce RTX 4070 Ti Super GPU. Individual runtimes are discussed in the corresponding sections.

3.1 Heat equation

Our first numerical example is the heat equation with homogeneous boundary conditions on the domain $\Omega = (0, 1)$ with final time $T = 1$, namely

$$\begin{aligned} \partial_\tau u - \alpha^2 \partial_{xx} u &= 0, && x \in (0,1),\ \tau \in (0,1) \\ u(x,0) &= f(x), && x \in (0,1) \\ u(0,\tau) = u(1,\tau) &= 0, && \tau \in (0,1). \end{aligned} \tag{3.3}$$

For the experiments, we choose $\alpha = 0.25$. As in [44, Sec. 3.5], we use initial conditions given by a truncated sine expansion, namely

$$u(x,0) = f(x) = \sum_{\ell=1}^{9} a_\ell \sin(\ell \pi x), \qquad x \in [0,1]$$

with coefficients $a_\ell \in [-1, 1], \ell \in \mathbb{N}$, which leads to the analytical solution

$$u(x,\tau) = \sum_{\ell=1}^{9} a_\ell \sin(\ell \pi x) \exp\left(-\ell^2 \pi^2 \alpha^2 \tau\right), \qquad x \in [0,1], \tau \in [0,1]. \tag{3.4}$$

Clearly, the truncation together with the condition $a_\ell \in [-1, 1]$ restricts the set of admissible initial conditions f. Functions far outside this class require a modified parameterization, for instance by increasing the number of retained modes the range of the coefficients. As explained in Remark 2.2, we shift both PDE variables x and t via the invertible transformation $\mathcal{T}\colon [-1, 1] \to [0, 1]$ with $\mathcal{T}(z) = \frac{1}{2}(z + 1)$ to be able to use the Chebyshev basis.

Remark 3.1. *Choosing multivariate Chebyshev polynomials as the underlying BOPB can be theoretically justified for several of our examples. For the heat equation, the explicit solution formula* (3.4) *shows that each input a_ℓ enters the solution only linearly, and only through a single summand. This linear dependence is reproduced exactly by the degree-one Chebyshev polynomial $T_1(a_\ell) = \sqrt{2}\, a_\ell$ (with $T_0 \equiv 1$ in the remaining coordinates), so that* (3.4) *may be rewritten as*

$$\begin{aligned} u(x,\tau,\boldsymbol{a}) &= \sum_{\ell=1}^{9} \frac{T_1(a_\ell)}{\sqrt{2}} \sin(\ell \pi x) \exp\big(-\ell^2 \pi^2 \alpha^2 \tau\big) \\ &= \sum_{\ell=1}^{9} \frac{T_{\boldsymbol{e}_\ell}(\boldsymbol{a})}{\sqrt{2}} \sin(\ell \pi x) \exp\big(-\ell^2 \pi^2 \alpha^2 \tau\big), \end{aligned}$$

where $\boldsymbol{e}_\ell = (0, \dots, 0, 1, 0, \dots, 0)^\top \in \mathbb{N}_0^9$ is the ℓ-th standard unit vector. Consequently, the relevant parameter dependence is concentrated on only the nine multi-indices $\boldsymbol{e}_1, \dots, \boldsymbol{e}_9$. One therefore expects a suitable index set I_{est} to consist of indices of the form $\boldsymbol{k} = (*, *, \boldsymbol{e}_\ell)^\top$, $\ell = 1, \dots, 9$, where the first two entries (the indices in the x- and τ-directions) remain arbitrary. Hence, the corresponding support is not only sparse but also highly structured.

For the sampling, we solve the PDE (3.3) using SciPy's `solve_ivp` with the implicit Radau IIA Runge–Kutta method of order five, a relative tolerance of 10^{-8}, and an absolute tolerance of 10^{-10}. In Algorithm 1, we use sparsity $s = 1000$ and extension $N = 64$ for the search space Γ, i.e., $\Gamma = [0, 64]^{11}$ and $|\Gamma| \approx 8.75 \cdot 10^{19}$.

The `cMR1L` comparison from [44, Sec. 3.5] uses the same parameters. As a learning-based baseline, we train a TFNO with 529 233 parameters on 10 000 and 40 000 training samples, respectively, resulting in 10 100 and 40 500 total PDE solves including validation data. We denote these approaches as $\texttt{TFNO}_{10k}$ and $\texttt{TFNO}_{40k}$, respectively.

Figure 3.1 illustrates the relative approximation error for all methods. `OMP+` achieves a median error of approximately $4 \cdot 10^{-5}$, comparable to `cMR1L`. This improves over `OMP`, which attains a median error of roughly $2 \cdot 10^{-4}$. The baselines $\texttt{TFNO}_{40k}$ and $\texttt{TFNO}_{10k}$ yield slightly larger median errors of about $3 \cdot 10^{-4}$ and $5 \cdot 10^{-4}$, respectively.

Figure 3.2 shows the number of solves, the corresponding computation times, and the remaining runtime excluding PDE solves. `OMP` required slightly more than 3 hours, with the dominant portion spent on solving the PDEs for sample generation. Similarly, `cMR1L` required roughly 700 hours due to the significantly larger number of required samples. The PDE-specific `OMP+` version was the fastest overall, with a total runtime of about 1 hour, roughly half of which was spent on solving the PDEs. The $\texttt{TFNO}_{10k}$ and $\texttt{TFNO}_{40k}$ required approximately 6 and 23 hours of training time, respectively.

The effect of the sparsity parameter s on the approximation quality is illustrated in Figure 3.3 for `OMP+` with extension $N = 64$. The relative error err_{rel} decreases as s increases, while the spread of errors remains within roughly one order of magnitude in each setting.

3.2 Burgers' equation

Our second example is the non-linear 1D Burgers' equation with homogeneous boundary conditions, namely

$$\begin{aligned} \partial_\tau u + u\partial_x u = \nu \partial_{xx} u, \qquad & x \in (0,1),\ \tau \in (0,1) \\ u(x,0) = f(x), \qquad & x \in (0,1) \\ u(0,\tau) = u(L,\tau) = 0, \qquad & \tau \in (0,1), \end{aligned}$$

which we solve for the viscosity parameter $\nu = 0.05$. In contrast to the previous example, we are only interested in the solution $u(\cdot, 1)$ at the final time. Again, given $\boldsymbol{a} \in [-1, 1]^9$, we consider initial conditions f given by the truncated sine expansion

$$f(x) \approx \sum_{\ell=1}^{9} a_\ell \sin(\ell \pi x), \qquad x \in [0, 1].$$

As before, we transform the spatial variable via $\mathcal{T}x = \frac{1}{2}(x + 1)$ and use `solve_ivp` with the Radau method, this time with a relative tolerance of 10^{-6} and an absolute tolerance of 10^{-8}. In Algorithm 1, we also use sparsity $s = 1000$ and extension $N = 64$. Since the target

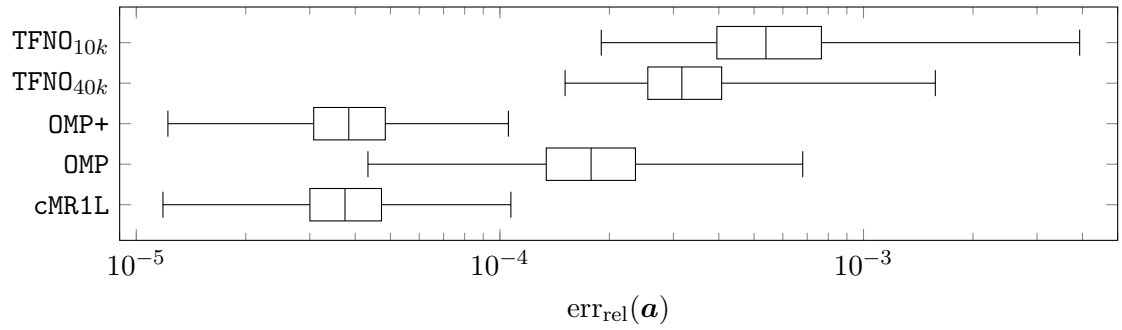


Figure 3.1: Relative approximation error $\mathrm{err}_{\mathrm{rel}}(\boldsymbol{a})$ for 10000 randomly drawn $\boldsymbol{a}$ for the heat equation. Box-and-whisker plots show median, first and third quartiles, and maximum and minimum errors.

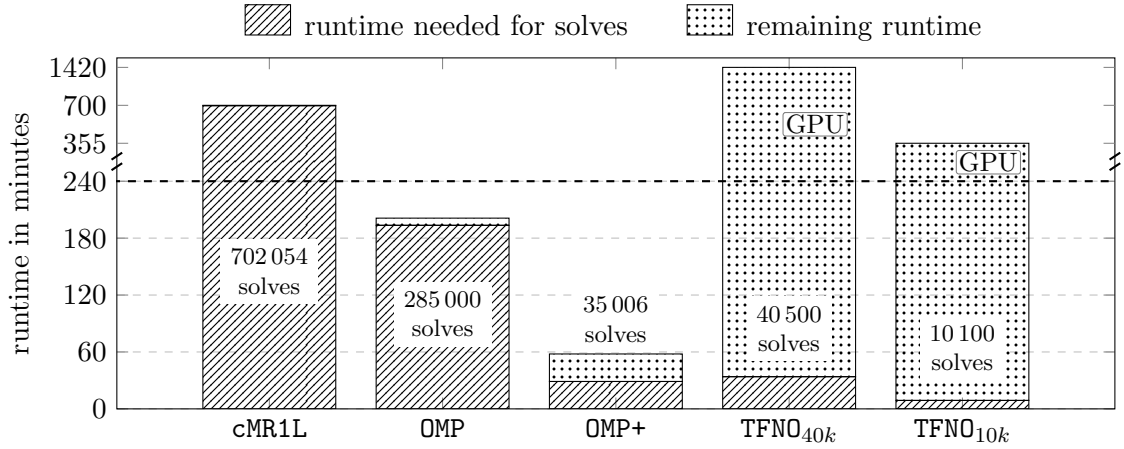


Figure 3.2: Runtime and sample complexity for the heat equation. The upper axis is compressed for visualization. Each bar is split into PDE solve time (lower) and remaining runtime (upper). All computations were performed on the same CPU, except for TFNO training, which used a GPU.

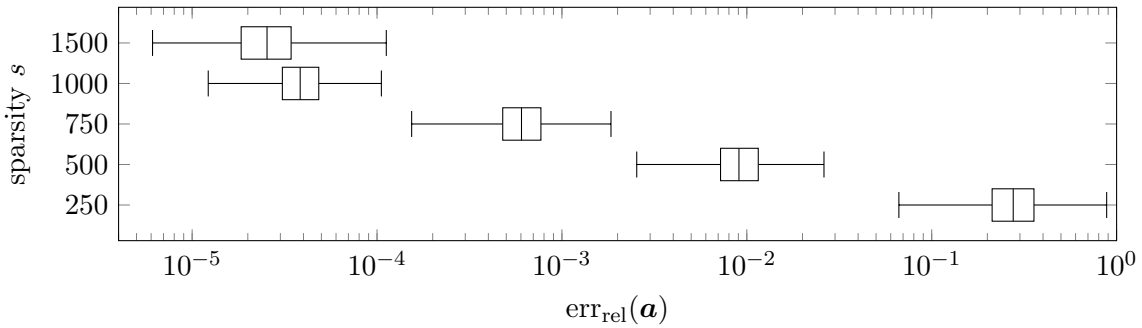


Figure 3.3: Relative approximation error $\mathrm{err}_{\mathrm{rel}}(\boldsymbol{a})$ for 10000 randomly drawn $\boldsymbol{a}$ for the heat equation using OMP+ with varying sparsity s. Box-and-whisker plots show median, first and third quartiles, and maximum and minimum errors.

function space is one-dimensional, `OMP+` would use only 3 space-time evaluations per PDE solve in the recommended default configuration. To improve the efficiency, we increase this to 9, giving 10 evaluations per solve, consistent with the previous example. Regarding the results in [44], already sparsity $s = 500$ and extension $N = 32$ lead to runtimes on the order of several days, making larger configurations infeasible. We highlight this mismatch in s by denoting the comparison as `cMR1L`*. The `TFNO` baseline with 81 049 parameters was trained on 40 000 samples, resulting in 40 500 total PDE solves including validation data.

Figure 3.4 shows the relative and absolute approximation errors for all approaches. The `TFNO` median error is almost two orders of magnitude smaller than those of the remaining approaches, which range between 10^{-2} and 10^{-1}. The relative error $\mathrm{err}_{\mathrm{rel}}(\boldsymbol{a})$ exhibits large upper whiskers for all approaches, which are absent for the absolute error $\mathrm{err}_{\mathrm{abs}}(\boldsymbol{a})$.

Figure 3.5 shows the computation time and number of PDE solves for each approach. `cMR1L`* required significantly more solves than the rest, resulting in substantially longer runtime. `OMP` and `TFNO` both finished in less than 5 hours, while `OMP+` needed only about 1 hour.

Lastly, Figure 3.6 depicts the absolute approximation error $\mathrm{err}_{\mathrm{abs}}(\boldsymbol{a})$ for `OMP+` with varying numbers of additional spatial evaluations per PDE solve (3, 9, and 15) and oversampling factors ($c \in \{1, 3\}$). The error magnitude remains similar across all six variations, with accuracy improving slightly for more spatial evaluations or higher oversampling factor c.

3.3 Parametric diffusion equation

As third example, we consider the 2D diffusion equation on $\Omega = [0, 1]^2$ with homogeneous Dirichlet boundary conditions and an affine random diffusion coefficient in some domain $\Omega_{\boldsymbol{y}}$, see [4, 3] namely

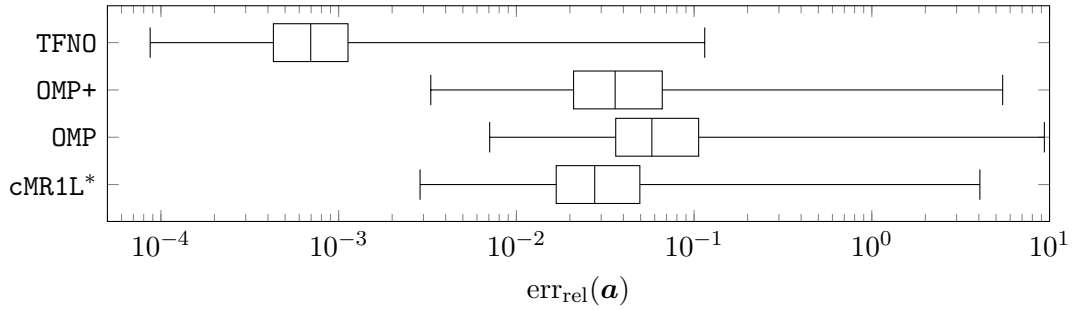
$$\begin{aligned} -\nabla \cdot (a(\boldsymbol{x}, \boldsymbol{y}) \nabla u(\boldsymbol{x}, \boldsymbol{y})) &= f(\boldsymbol{x}), & \boldsymbol{x} \in \Omega,\ \boldsymbol{y} \in \Omega_{\boldsymbol{y}}, \\ u(\boldsymbol{x}, \boldsymbol{y}) &= 0, & \boldsymbol{x} \in \partial\Omega,\ \boldsymbol{y} \in \Omega_{\boldsymbol{y}}. \end{aligned} \tag{3.5}$$

Here, the differential operator ∇ acts with respect to the spatial variable $\boldsymbol{x}$. As in [15, Sec. 11] and [22, Sec. 4.3], we use the affine coefficient

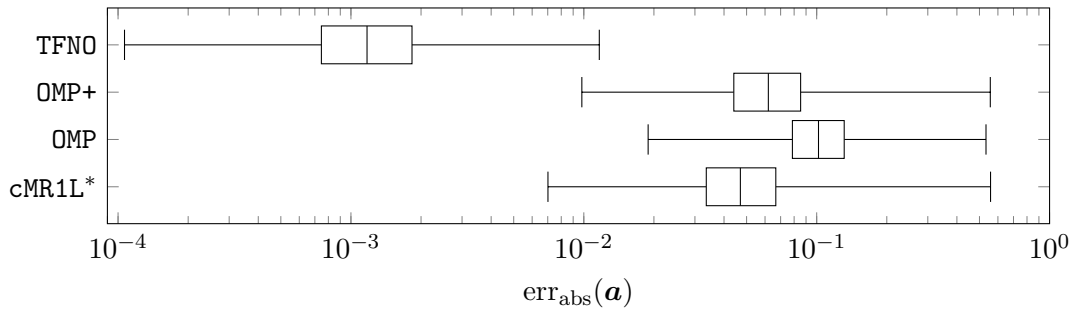
$$a(\boldsymbol{x}, \boldsymbol{y}) = 1 + \sum_{j=1}^{20} y_j \psi_j(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega,\ \boldsymbol{y} \in [-1, 1]^{20},$$

with $\psi_j(\boldsymbol{x}) = cj^{-\mu} \cos(2\pi m_1(j) x_1) \cos(2\pi m_2(j) x_2)$. The parameter are chosen as $c = 0.9/\zeta(2)$ and $\mu = 2$, together with $k(j) \coloneqq \lfloor -1/2 + \sqrt{1/4 + 2j} \rfloor$, $m_1(j) \coloneqq j - \frac{k(j)(k(j)+1)}{2}$ and $m_2(j) \coloneqq k(j) - m_1(j)$. Moreover, we set $f \equiv 1$. As before, we transform $\boldsymbol{x}$ componentwise by the affine map $\mathcal{T}(z) = \frac{1}{2}(z + 1)$. In total, this yields a 22-dimensional approximation problem.

As in [22], we solve the PDE (3.5) numerically using FEniCS on a mesh with 5000 finite elements. In Algorithm 1, we choose $s = 1000$ and $N = 64$. Moreover, we restrict the candidate set Γ by imposing $\max_{\boldsymbol{k} \in \Gamma} \|\boldsymbol{k}\|_0 = 7$. Such a restriction reflects the common observation that, in many high-dimensional parametric PDEs, higher-order interactions contribute considerably less than lower-order ones. This choice is adopted from [44, Sec. 3.4] to ensure a comparable experimental setup. The `cMR1L` comparison uses the same parameters. The `TFNO` with 529 233 parameters is trained on 85 000 samples, resulting in a total of 86 000 PDE solves including validation data.



(a) The relative approximation error $\mathrm{err}_{\mathrm{rel}}(\boldsymbol{a})$.



(b) The absolute approximation error $\mathrm{err}_{\mathrm{abs}}(\boldsymbol{a})$.

Figure 3.4: The relative and absolute approximation error $\mathrm{err}(\boldsymbol{a})$ for 10000 randomly drawn $\boldsymbol{a}$ for the Burgers' equation example. The box-and-whisker plots show the median, the first and the third quartile as well as the maximal and minimal error observed. Note that cMR1L* uses a smaller sparsity s and extension N.

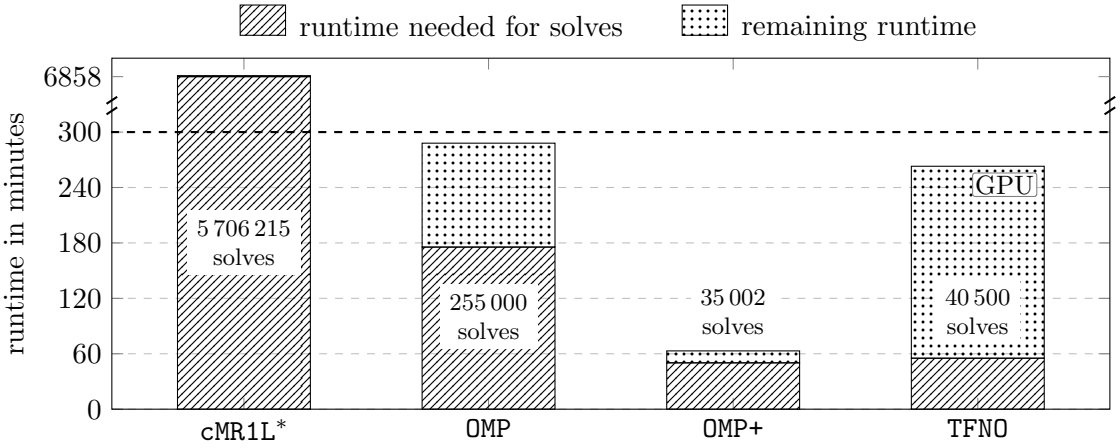


Figure 3.5: Runtime and sample complexity for the Burgers' equation. The upper axis is compressed for visualization. Each bar is split into PDE solve time (lower) and remaining runtime (upper). All computations were performed on the same CPU, except for TFNO training, which used a GPU.

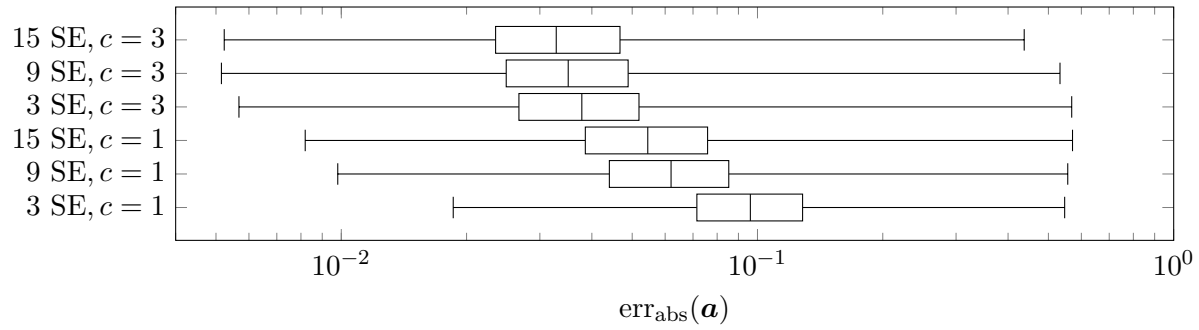


Figure 3.6: Absolute approximation error $\mathrm{err}(\boldsymbol{a})$ for 10000 randomly drawn $\boldsymbol{a}$ for the Burgers' equation using `OMP+` with sparsity $s = 1000$, varying numbers of additional spatial evaluations (SE) and oversampling factors c. Box-and-whisker plots show median, first and third quartiles, and maximum and minimum errors.

Figure 3.7 compares the relative approximation error $\mathrm{err}(\boldsymbol{y})$ for all four approaches. The `TFNO` achieves a median error of roughly $2.5 \cdot 10^{-4}$, almost an order of magnitude below the other approaches, whose median errors are around 10^{-3}. On the other hand, the `TFNO`'s upper whisker spans almost a full order of magnitude, compared to roughly half an order of magnitude for the other approaches.

Figure 3.8 shows the number of PDE solves and corresponding runtimes for all approaches. As in the previous example, `cMR1L` required significantly more solves and runtime than `OMP` and `OMP+`, which finished in roughly 8 and 2 hours, respectively. Notably, the `TFNO` training time is comparable to the total runtime of `cMR1L`.

3.4 Discussion

Now, we discuss the numerical results and the observed trade-offs between approximation quality and computational effort.

Sparse approximation A key observation is that the approximation accuracy of `OMP` and `OMP+` depends primarily on the sparsity of the solution u in the chosen basis. For the heat equation, u is highly sparse in the multivariate Chebyshev basis due to the linear dependence on the parameters a_ℓ in (3.4). Consequently, the index set I_{est} detected by Algorithm 1 already performs well for small sparsity $s = |I_{\mathrm{est}}|$, and increasing the sparsity s further improves accuracy, as seen in Figure 3.3. The increase saturates once the diameter of the search space Γ (which is restricted by N) is exhausted.

The Burgers' equation, by contrast, showed the opposite behavior: u is not well represented by sparse Chebyshev expansions, as discussed in [44, Sec. 3.6]. This results in much lower approximation accuracy at the same sparsity s. Moreover, increasing the amount of training data improves accuracy only marginally, as seen in Figure 3.6, suggesting that the observed limitations stem from the basis rather than insufficient data for the sparse recovery step.

These results highlight that the choice of basis critically affects the approximation quality. When the detected index set I_{est} exhibits little sparsity structure, experimenting with alternative bases is advisable. Indeed, `OMP` and `OMP+` are applicable for arbitrary product bases, provided that an efficient routine for the matrix-vector multiplication with the basis matrix (2.12) is available.

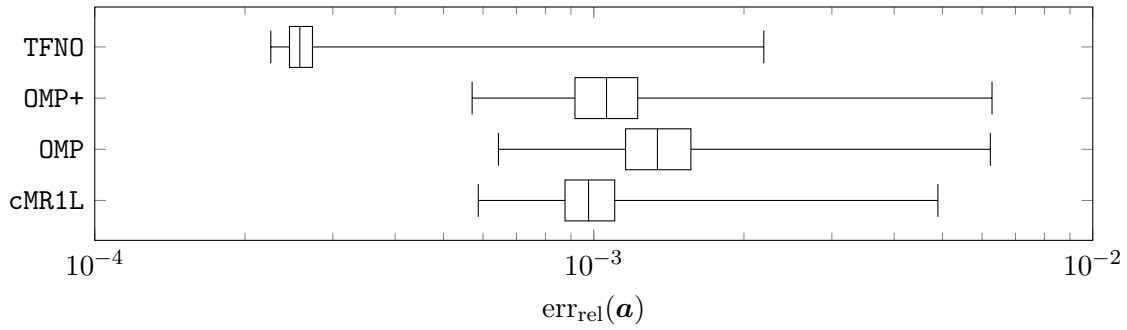


Figure 3.7: Relative approximation error $\mathrm{err}_{\mathrm{rel}}(\boldsymbol{y})$ for 10000 randomly drawn $\boldsymbol{y}$ for the parametric diffusion equation. Box-and-whisker plots show median, first and third quartiles, and maximum and minimum errors.

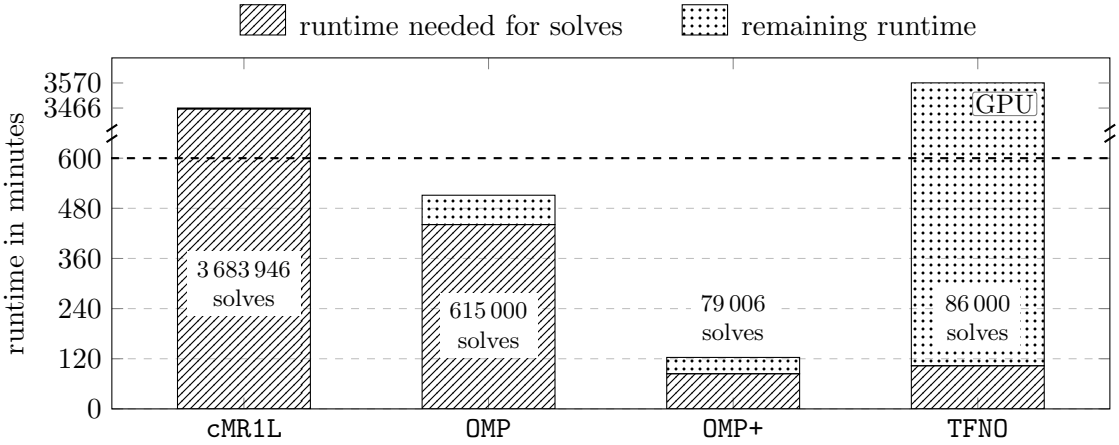


Figure 3.8: Runtime and sample complexity for the parametric diffusion equation. The upper axis is compressed for visualization. Each bar is split into PDE solve time (lower) and remaining runtime (upper). All computations were performed on the same CPU, except for `TFNO training`, which used a GPU.

Samples and PDE solves Applying sparse recovery methods such as OMP to the dimension-incremental framework yields a substantial advantage in sample complexity over cubature approaches based on rank-1 lattices. While the rank-1 lattice methods in [44] enable efficient reconstruction via FFT, this benefit is outweighed by the large number of required PDE solves, which scales as s^2 in the worst case. In contrast, the number of samples required by `OMP` scales only linearly in s. Furthermore, the additional samples per PDE solve in `OMP+` (which are not usable for cubature-based methods) yield a substantial additional reduction in computational cost (by up to 80%) without sacrificing approximation quality, clearly visible across all experiments. As a consequence, our methods close the gap in terms of sample complexity to learning-based approaches such as TFNOs, though the dimension-incremental framework does require samples with a particular structure.

Moreover, we want to emphasize that the `TFNO` models serve purely as baselines, following the generic guidelines of the neural operator library [27] without further modification. The

comparison should therefore not be interpreted as a rigorous benchmark, but rather as an indication of the approximation quality and computational complexity achievable by our methods relative to off-the-shelf neural operator configurations.

Interpretability of the approximation An important advantage of the proposed approach is its interpretability. The detected index set I_{est} reveals which input variables are most influential, which parameter interactions are relevant, and to what extent higher-order interactions matter. The resulting approximation is thus not merely a black-box predictor, but also yields qualitative insight into the underlying PDE. In contrast, such structural information is far less accessible in neural operator approaches such as TFNOs, where the learned representation is distributed across a large number of trainable parameters.

References

[1] K. Azizzadenesheli, N. Kovachki, Z. Li, M. Liu-Schiaffini, J. Kossaifi, and A. Anandkumar. Neural operators for accelerating scientific simulations and design. *Nat. Rev. Phys.*, 6:320–328, 2024.

[2] M. Bachmayr, A. Cohen, and W. Dahmen. Parametric PDEs: Sparse or low-rank approximations? *IMA J. Numer. Anal.*, 38:1661–1708, 2018.

[3] M. Bachmayr, A. Cohen, R. DeVore, and G. Migliorati. Sparse polynomial approximation of parametric elliptic PDEs. Part II: lognormal coefficients. *ESAIM Math. Model. Numer. Anal.*, 51(1):341–363, 2017.

[4] M. Bachmayr, A. Cohen, and G. Migliorati. Representations of Gaussian random fields and approximation of elliptic PDEs with lognormal coefficients. *J. Fourier Anal. Appl.*, 24:621 – 649, 2016.

[5] F. Bartolucci, E. de Bezenac, B. Raonic, R. Molinaro, S. Mishra, and R. Alaifari. Representation equivalent neural operators: a framework for alias-free operator learning. In *37th Conference on Neural Information Processing Systems*, 2023.

[6] P. Batlle, M. Darcy, B. Hosseini, and H. Owhadi. Kernel methods are competitive for operator learning. *J. Comput. Phys.*, 496:112549, 2024.

[7] A. Belloni, V. Chernozhukov, and L. Wang. Square-root LASSO: Pivotal recovery of sparse signals via conic programming. *Biometrika*, 98(4):791–806, 2011.

[8] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart. Model reduction and neural networks for parametric PDEs. *SMAI J. Comput. Math.*, 7:121–157, 2021.

[9] N. Boullé, C. J. Earls, and A. Townsend. Data-driven discovery of Green's functions with human-understandable deep learning. *Sci. Rep.*, 12:4824, 2022.

[10] N. Boullé and A. Townsend. A mathematical guide to operator learning. In *Numerical Analysis Meets Machine Learning*, volume 25 of *Handbook of Numerical Analysis*, pages 83–125. Elsevier, 2024.

[11] A. Chkifa, A. Cohen, and C. Schwab. High-dimensional adaptive sparse polynomial interpolation and applications to parametric PDEs. *Found. Comput. Math.*, 14(4):601–633, 2014.

[12] D. Dũng, V. N. Temlyakov, and T. Ullrich. *Hyperbolic Cross Approximation*. Advanced Courses in Mathematics – CRM Barcelona. Birkhäuser, Cham, 2018.

[13] M. V. de Hoop, N. B. Kovachki, N. H. Nelsen, and A. M. Stuart. Convergence rates for learning linear operators from noisy data. *SIAM/ASA J. Uncertain. Quantif.*, 11(2):480–513, 2023.

[14] V. Duruisseaux, J. Kossaifi, and A. Anandkumar. Fourier neural operators explained: A practical perspective. *arXiv:2512.01421*, 2025.

[15] M. Eigel, C. J. Gittelson, C. Schwab, and E. Zander. Adaptive stochastic Galerkin FEM. *Comput. Methods Appl. Mech. Engrg.*, 270:247–269, 2014.

[16] V. S. Fanaskov and I. V. Oseledets. Spectral neural operators. *Dokl. Math.*, 108:226–232, 2023.

[17] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Birkhäuser/Springer, New York, 2013.

[18] Z. Hao, Z. Wang, H. Su, C. Ying, Y. Dong, S. Liu, Z. Cheng, J. Song, and J. Zhu. GNOT: A general neural operator transformer for operator learning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*. PMLR, 2023.

[19] J. Hertrich and S. Neumayer. Generative feature training of thin 2-layer networks. *Trans. Mach. Learn. Res.*, 2025.

[20] T. Ingebrand, A. J. Thorpe, S. Goswami, K. Kumar, and U. Topcu. Basis-to-basis operator learning using function encoders. *Comput. Methods Appl. Mech. Engrg.*, 435:117646, 2025.

[21] H. Kadri, E. Duflos, P. Preux, S. Canu, A. Rakotomamonjy, and J. Audiffren. Operator-valued kernels for learning from functional response data. *J. Mach. Learn. Res.*, 17:1–54, 2016.

[22] L. Kämmerer, D. Potts, and F. Taubert. The uniform sparse FFT with application to PDEs with random coefficients. *Sampl. Theory Signal Process. Data Anal.*, 20:19, 2022.

[23] L. Kämmerer, D. Potts, and F. Taubert. Nonlinear approximation in bounded orthonormal product bases. *Sampl. Theory Signal Process. Data Anal.*, 21:19, 2023.

[24] L. Kämmerer, D. Potts, and T. Volkmer. High-dimensional sparse FFT based on sampling along multiple rank-1 lattices. *Appl. Comput. Harmon. Anal.*, 51:225–257, 2021.

[25] L. Kämmerer, T. Ullrich, and T. Volkmer. Worst case recovery guarantees for least squares approximation using random samples. *Constr. Approx.*, 54:295–352, 2021.

[26] G. Kissas, G. J. Pappas, P. Perdikaris, and J. Seidman. NOMAD: Nonlinear manifold decoders for operator learning. In *Advances in Neural Information Processing Systems 35*, Advances in Neural Information Processing Systems 35, pages 5601–5613. Curran Associates, Inc., 2022.

[27] J. Kossaifi, N. Kovachki, Z. Li, D. Pitt, M. Liu-Schiaffini, R. J. George, B. Bonev, K. Azizzadenesheli, J. Berner, and A. Anandkumar. NeuralOperator: Learning in infinite dimensions with neural operators (Python implementation), 2025. GitHub repository, Link: https://github.com/neuraloperator/neuraloperator.

[28] J. Kossaifi, N. B. Kovachki, K. Azizzadenesheli, and A. Anandkumar. Multi-grid tensorized Fourier neural operator for high-resolution PDEs. *Trans. Mach. Learn. Res.*, 2024.

[29] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *J. Mach. Learn. Res.*, 24(89):1–97, 2023.

[30] N. B. Kovachki, S. Lanthaler, and H. Mhaskar. Data complexity estimates for operator learning. *J. Mach. Learn. Res.*, 2026.

[31] N. B. Kovachki, S. Lanthaler, and A. M. Stuart. Operator learning: Algorithms and analysis. In *Numerical Analysis Meets Machine Learning*, volume 25 of *Handbook of Numerical Analysis*, pages 419–467. Elsevier, 2024.

[32] S. Kunis and H. Rauhut. Random sampling of sparse trigonometric polynomials. II. Orthogonal matching pursuit versus basis pursuit. *Found. Comput. Math.*, 8(6):737–763, 2008.

[33] S. Lanthaler, S. Mishra, and G. E. Karniadakis. Error estimates for DeepONets: A deep learning framework in infinite dimensions. *Trans. Math. Appl.*, 6(1):tnac001, 2022.

[34] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[35] Z. Li, K. Meidani, and A. B. Farimani. Transformer for partial differential equations' operator learning. *Trans. Mach. Learn. Res.*, 2023.

[36] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/IMS J. Data Sci.*, 1(3):1–27, 2024.

[37] M. Liu-Schiaffini, J. Berner, B. Bonev, T. Kurth, K. Azizzadenesheli, and A. Anandkumar. Neural operators with localized integral and differential kernels. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

[38] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nat. Mach. Intell.*, 3(3):218–229, 2021.

[39] M. Moeller, S. Neumayer, K. Pozharska, T. Sommerfeld, and T. Ullrich. High-dimensional sparse recovery from function samples: Decoders, guarantees, and instance optimality. *arXiv:2503.16209*, 2026.

[40] C. Mora, A. Yousefpour, S. Hosseinmardi, H. Owhadi, and R. Bostanabad. Operator learning with Gaussian processes. *Comput. Methods Appl. Mech. Engrg.*, 434:117581, 2025.

[41] D. Needell and J. A. Tropp. CoSaMP: iterative signal recovery from incomplete and inaccurate samples. *Appl. Comput. Harmon. Anal.*, 26(3):301–321, 2009.

[42] N. H. Nelsen and A. M. Stuart. Operator learning using random features: A tool for scientific computing. *SIAM Rev.*, 66(3):535–571, 2024.

[43] Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pages 40–44 vol.1, 1993.

[44] D. Potts and F. Taubert. An approach to discrete operator learning based on sparse high-dimensional approximation. *Electron. Trans. Numer. Anal.*, 63:468–495, 2025.

[45] D. Potts and T. Volkmer. Sparse high-dimensional FFT based on rank-1 lattice sampling. *Appl. Comput. Harmon. Anal.*, 41(3):713–748, 2016.

[46] H. Rauhut and R. Ward. Sparse Legendre expansions via ℓ_1-minimization. *J. Approx. Theory*, 164:517–533, 2012.

[47] B. L. T. Sturm and M. G. Christensen. Comparison of orthogonal matching pursuit implementations. In *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, pages 220–224, 2012.

[48] U. Subedi and A. Tewari. Operator learning: A statistical perspective. *Annu. Rev. Stat. Appl.*, 13:123–148, 2026.

[49] J. Sun, Y. Liu, Z. Zhang, and H. Schaeffer. Towards a foundation model for partial differential equations: Multioperator learning and extrapolation. *Phys. Rev. E*, 111:035304, 2025.

[50] F. Taubert. NABOPB: Nonlinear approximation in bounded orthonormal product bases (Python implementation), 2026. Version: v2.0.0, Contributors: L. Kämmerer and S. Neumayer, Link: https://doi.org/10.5281/zenodo.20522832.

[51] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. Inform. Theory*, 53(12):4655–4666, 2007.

[52] S. Wang, H. Wang, and P. Perdikaris. Improved architectures and training algorithms for deep operator networks. *J. Sci. Comput.*, 92(2), 2022.

[53] T. Zhang. Sparse recovery with orthogonal matching pursuit under RIP. *IEEE Trans. Inform. Theory*, 57(9):6215–6221, 2011.