EPJ Nuclear Sci. Technol. **11**, 61 (2025)

https://doi.org/10.1051/epjn/2025055

EPJ Nuclear Sciences & Technologies

Special Issue on 'Overview of recent advances in HPC simulation methods for nuclear applications', edited by Andrea Zoia, Elie Saikali, Cheikh Diop and Cyrille de Saint Jean

Available online at: https://www.epj-n.org

REGULAR ARTICLE

OPEN ACCESS

Multi-output Gaussian processes for the reconstruction of homogenized cross-sections

Olivier Truffinet[1,2,*,a], Karim Ammar[2], Jean-Philippe Argaud[1], Nicolas Gérard Castaing[2], and Bertrand Bouriquet[3]

[1] EDF R&D PERICLES, 7 Boulevard Gaspard Monge, 91120 Palaiseau, France
[2] Université Paris-Saclay, CEA, Service d'Études des Réacteurs et de Mathématiques Appliquées, 91190 Gif-sur-Yvette, France
[3] EDF DQI, 2 Rue Ampère, 93206 Saint-Denis Cedex, France

Received: 1 June 2025 / Received in final form: 6 August 2025 / Accepted: 7 August 2025

Abstract. Deterministic nuclear reactor neutronics codes employing the prevalent two-step scheme often generate a substantial amount of intermediate data at the interface of their two subcodes, which can impede the overall performance of the software. The bulk of this data comprises "few-groups homogenized cross-sections" or HXS, which are stored as tabulated multivariate functions and interpolated inside the core code. A number of mathematical tools have been studied for this interpolation purpose over the years, but few meet all the challenging requirements of neutronics computation chains: extreme accuracy, low memory footprint, fast predictions. We here present a new framework to tackle this task, based on multi-output Gaussian processes (MOGP). These smooth and tunable bayesian regressors are able to model several quantities at once, and to capture the correlations between them – a key asset in the modeling of HXS's, which we show to be highly similar from one another. Several models of this family are discussed, compared, adapted to the case of very numerous HXS's, and their possible modeling choices are experimented on. These machine learning models enable us to interpolate HXS's with improved accuracy compared to the current multilinear standard, using only a fraction of its training data – meaning that the amount of required precomputation is reduced by a factor of several dozens. They also necessitate an even smaller fraction of its storage requirements, preserve its reconstruction speed, and unlock new functionalities such as adaptive sampling and facilitated uncertainty quantification. We demonstrate the efficiency of this approach on a rich test case reproducing the VERA benchmark, proving in particular its scalability to datasets of millions of HXS's.

1 Introduction

1.1 Problem description

Most deterministic neutronics codes make use of the so-called "two-step scheme", and thus pass processed nuclear data at the interface of their two subcodes (lattice code and core code). This data consists mainly of **few-groups homogenized cross-sections (denoted HXS in this article)**, which quantify the interaction of neutrons with matter. For a given isotope i and nuclear reaction r, a spatial region V and an energy range $[E_{g-1}; E_g]$, the corresponding HXS is defined as:

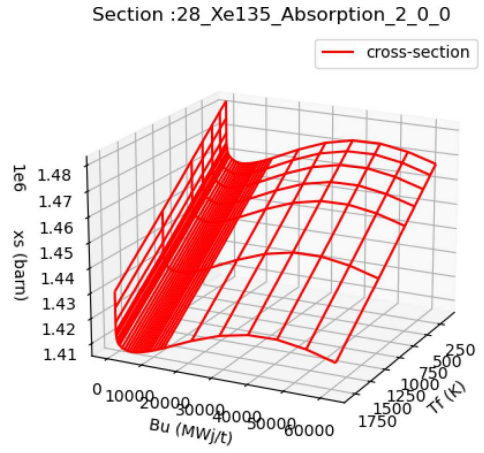
$$\sigma_{i,r}^{g,V}(\mathbf{P}) = \frac{\displaystyle\int_{\substack{E_{g-1}\leq E\leq E_g \\ \mathbf{r}\in V}} \sigma_{i,r}(E,T)\,\Phi(\mathbf{r},E,\mathbf{P})\,d\mathbf{r}dE}{\displaystyle\int_{\substack{E_{g-1}\leq E\leq E_g \\ \mathbf{r}\in V}} \Phi(\mathbf{r},E,\mathbf{P})\,d\mathbf{r}dE} \quad (1)$$

where $\sigma_{i,r}(E,T)$ is the native, "physical" cross-section as delivered by a neutronics data library, Φ the (scalar) neutron flux, and $\mathbf{P}$ a set of operating parameters (temperatures, burnup, etc.) on which the flux Φ computed by the lattice code depends. The flux-weighted averaging aims at preserving reaction rates in the given energy range; this definition makes the HXS's depend on the operating parameters $\mathbf{P}$. They are therefore represented as multivariate functions, of generally 3 to 7 variables[1].

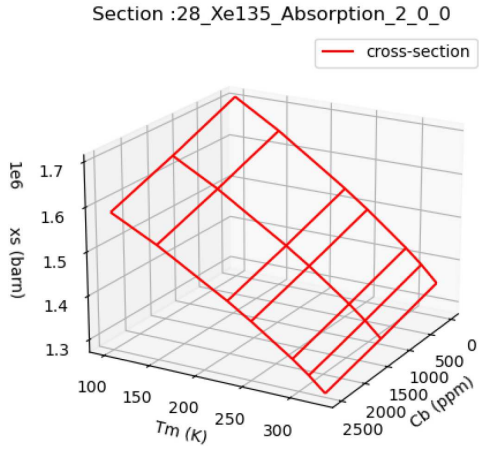
* e-mail: olivier.truffinet@edf.fr

[a] Currently employed at institution 1, but worked at institution 2 at the time of this work.

[1] Because of definition (1), these variables can be macroscopic and global (boron concentration in the moderator, assembly-level burnup...) even if the notion of cross-section is intuitively attached to an isolated nucleus.



(a)



(b)

Fig. 1. Typical aspect of the 2-group thermal absorption XHS of Xenon 135 (homogenized on a fuel cell). The wire-frame mesh corresponds to a realistic multivariate grid for multilinear interpolation. (a) Projection on the Bu (burnup) and T_f (fuel temperature) variables. (b) Projection on the T_m (moderator temperature) and C_b (boron concentration) variables.

To solve a given reactor state, the core solver requires a set of HXS matching the current operating conditions; but these cross-sections can usually not be computed on-the-fly, lest computation times would skyrocket. In current computation chains, they are therefore tabulated in advance, then interpolated inside the core code. A popular tool for this has been *multilinear interpolation* [1,2], which is simply a piecewise interpolation method of degree 1; but despite some significant advantages (in particular simplicity, error control and speed), this model is sub-optimal in terms of accuracy, data usage efficiency and storage requirements, which is becoming more and more of a burden as the finesse of simulations increases. There is thus a pressing need to develop more efficient interpolation/regression schemes, by exploring three complementary approaches:

- **Finding better function approximators**: HXS's are very regular quantities, so it is questionable to approximate them with local, memory-based methods such as multilinear interpolation when proficient and easily-tunable smooth estimators are available;
- **Getting rid of full grids**: HXS's exhibit weak interactions between variables, so fitting approximation models to them on full grids is very data-inefficient, and sparse or unstructured supports should be used instead;
- **Leveraging redundancy**: HXS's are strongly correlated between them, as most of their variations originate from the homogenization operation, performed with a flux which is common to all. Exploiting the low effective dimensionality of HXS datasets can thus be a powerful way of avoiding useless computation.

These three lines of work have been investigated in the last decades; note that the first two are often intertwined, as the choice of an approximator can put constraints on the interpolation support.

1.2 Existing methodologies

Better and grid-less approximators. *Polynomial models*, whether global or local, have been especially studied by practitioners. Works on HXS reconstruction with global polynomials include multivariate polynomial regression in [3] (backed by a cubic spline for the burnup variable), quasi-regression performed as an ANOVA analysis in [4], and hierarchical interpolation on sparse grids in [5]. Local polynomials refer to *splines*, which have been studied thoroughly in [6]. All these methods yielded satisfactory results, with relative interpolation errors of the order of 0.01%[2], and significant data and storage savings – especially with global models for which a few dozens of polynomial terms are usually enough to model a HXS. Yet they all have their limitations:

- They struggle with the build-up of many isotopic concentrations at the start of the reactor, which causes a sharp variation along the burnup axis in many HXS's[3] – as it can be seen in Figure 1a – and require some specific "trick" to account for it (addition of a cubic spline in burnup in [3], refined meshes at low Bu values in [6], etc.);
- They sometimes suffer from oscillation problems [6] – the well-known Runge phenomenon for polynomial interpolation [7];
- They still require large numbers of data samples to be fitted on, from 640 in [3] to dozens of thousands in [4];

[2] This seemingly impressive value must be put in perspective: HXS's often vary little around their mean value, which makes *all* relative errors low.

[3] This is due to the role of concentrations in *self-shielding* calculations, which play an important part in HXS generation by the lattice code.

- Some of them maintain an inconvenient grid-like structure: full grids for the spline models in [6], sparse grids in [5], factorized (burnup × other variables) support in [3] (which imposes to store and use *depletion cross-sections* in addition to the branch ones);
- Some of them rely heavily on hand-tuned hyperparameters, such as the location of spline knots in [6] or the anisotropy vector in [5], which could lead to problems of generality when applied to different datasets.

The first two items highlight a more general fact: low-order polynomial methods are known to struggle with steep or localized variations, so it is likely that future test-cases will put these methods in difficulty, in particular if more variables are added.

By contrast, *kernel methods* are in principle able to deal with any function shape, and work with any multidimensional set of points. First results have been obtained with them: in [8], Support Vector Regression was investigated and yielded good performance, with relative errors of the order of 0.05% – although with an unspecified number of training data. In [6], the author experimented on kernel regression and also obtained good results, reaching accuracies one order of magnitude higher than with multilinear interpolation for a same number of datapoints. Yet these two references don't leverage the full potential of kernel-based approximation: they still use Cartesian grids as an interpolation support, and use rather outdated regressors, devoid of automatic hyperparameter tuning in particular. Moreover, with these models the time spent to make a prediction is *linear in the number of training points*: this is less favorable than with multilinear interpolation, for which this duration is only proportional to 2^d with d the number of variables, and can turn out to be a problem if high interpolation speed is required in applications. Therefore, there is still room for improvement in this line of work.

Lastly, *Artificial Neural Networks* (ANNs) are a trending tool for many regression problems, and they are beginning to be investigated for the approximation of HXS's. Still in [6], the author experimented on many different ANN architectures, but didn't manage to reach the accuracy of multilinear interpolation, let alone this of his previous work on kernel methods. In [9], a three-layered multi-layer perceptron (MLP) – one model outputting the 20 HXS's contained in the dataset – was able to outperform multilinear interpolation, but at the cost of a 20-fold *increase* in storage requirements[4], and of a rather high prediction duration of $\simeq 10^{-4}$ s per HXS and point. In [10], the authors also used a (smaller) 3-layered multi-output MLP, coupled with a step of Principal Component Analysis (PCA) for dimensionality reduction, to simultaneously approximate all HXS's; they obtained good results, with relative errors of the order of 0.1%. The great originality of their approach is to get rid of the burnup variable by taking isotopic concentrations as inputs of the model, which also enables it to account for history effects. Lastly, the ANN of [11] uses a convolutional structure to model the spatial relationships between the equivalence parameters of all the fuel cells in a given assembly. In all these works, ANNs appear less promising than kernel methods, requiring more data to reach a similar accuracy, and making for less efficient computations; this will be discussed in the concluding section. They are also dependent on a profusion of hyperparameters, in particular the number of layers and the size of each one, which significantly affect the predictive performance and thus require systematical testing of many architectures.

Dimensionality reduction. As stated before, as computational data becomes more numerous in neutronics codes, it is also likely to become more redundant; by resorting to dimensionality reduction techniques (compression, projections...), one could therefore deal with this data more efficiently. Recent studies have begun exploring this avenue. In [12], singular values decomposition (SVD) was used to perform lossy compression on HXS's from distinct fuel cells; this method authorized 90% of memory savings at the expense of losing only 10^{-6} of the dataset's variance, which incurs only a few pcm of error on the k_{eff}. The same approach was followed in [13] with Boiling Water Reactor HXS's, with similar results. In [6] and [9], as previously mentioned, multi-output ANNs were able to jointly approximate all HXS's from a dataset with relatively low errors; these networks being both shallow (3 layers) and narrow (only ∼10 "neurons" in some layers), they constitute a low-dimensional representation of HXS datasets. Dimensionality reduction has also been used on other neutronic quantities: 618-group HXS-flux products in [14], isotopic concentration data in [15], or even local burnup data in [16]. These works are already showing the potential of dimensionality reduction in neutronic computation chains, although they could still be improved in several ways: better choices of variables to compress or neglect, addition of normalization schemes.

2 Description of the model

2.1 Single-output Gaussian processes

The basic theory of Gaussian processes (GP's) is now well-known in many applied mathematics communities; a good pedagogical introduction can be found in a reference manual on the topic [17]. As stated in definition 2.1 of this book, "*A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution*". Performing Gaussian process regression of some data starts by modeling these data points as observed (probabilistic) samples from an underlying Gaussian process[5]. Using the joint distribution of these observed values and unobserved ones, probabilities can be estimated for the quantity to be interpolated by *conditioning* this distribution on known (observed) values.

[4] This increase is not discussed in the paper; we computed it by comparing the number of weights of the proposed ANN and the number of grid points used by the multilinear baseline.

[5] This apparently intimidating modeling in fact makes no assumption about the data, apart from being continuous. Gaussian processes are known to be *universal approximators*: they can model any smooth function up to arbitrary precision if enough samples are provided.

To fix notations, let $(\mathbf{x}_i, y_i)_{1\leq i\leq n}$ be a set of n input-output datapoints, where the $\mathbf{x}_i$'s are d-dimensional and y_i's are scalars. The output values are supposed to be noisy observations of the interpolated quantity $f(\mathbf{x})$: $y = f(\mathbf{x}) + \epsilon(\mathbf{x})$, where ϵ is an i.i.d. (Independent and Identically Distributed) Gaussian noise of magnitude σ^2, that is, $\mathbb{E}[\epsilon(\mathbf{x})] = 0\ \forall \mathbf{x}$ and $\mathrm{Cov}[\epsilon(\mathbf{x}), \epsilon(\mathbf{x}')] = \sigma^2\delta_{\mathbf{x},\mathbf{x}'}$ (δ is the Kronecker symbol). We assume that f follows a Gaussian process of mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$, where $m(\cdot)$ can be any real function and $k(\cdot,\cdot)$ is a *kernel*, i.e a symmetric positive definite function. As for any Gaussian variable, f is entirely specified by its mean and covariance. If $\mathbf{x}_*$ is an unobserved input point, $\mathbf{X} = (\mathbf{x}_i)_{1\leq i\leq n}$ denotes the training data inputs and $\mathbf{y} = (y_i)_{1\leq i\leq n}$ their matching outputs, following the Gaussian process assumption, one can write the joint distribution of the unobserved and observed values:

$$\begin{bmatrix}\mathbf{y}\\ f(\mathbf{x}_*)\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}m(\mathbf{y})\\ m(f(\mathbf{x}_*))\end{bmatrix}, \begin{bmatrix}k(\mathbf{X},\mathbf{X}) + \sigma^2\mathbf{I} & k(\mathbf{X},\mathbf{x}_*)\\ k(\mathbf{X},\mathbf{x}_*)^T & k(\mathbf{x}_*,\mathbf{x}_*)\end{bmatrix}\right) \tag{2}$$

In order to simplify notations, we denote $\mathbf{K} = k(\mathbf{X},\mathbf{X})$, $\mathbf{k}_{\mathbf{x}_*\mathbf{X}} = k(\mathbf{X},\mathbf{x}_*)$ and $k_{x_*x_*} = k(\mathbf{x}_*,\mathbf{x}_*)$. One can deduce from Equation (2) the conditional distribution of $f(\mathbf{x}_*)$ given the observations (Eq. (3)); the mean of this Gaussian distribution constitutes the regression estimator *per se*, whereas its variance serves as a measure of uncertainty.

$$f(\mathbf{x}_*)|\mathbf{X},\mathbf{y} \sim \mathcal{N}(m(\mathbf{x}_*) + \mathbf{k}_{\mathbf{x}_*\mathbf{X}}^{\mathbf{T}}\left(\mathbf{K} + \sigma^2\mathbf{I}\right)^{-1}(\mathbf{y} - m(\mathbf{X})),\\ k_{x_*x_*} - \mathbf{k}_{\mathbf{x}_*\mathbf{X}}^{\mathbf{T}}\left(\mathbf{K} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{k}_{\mathbf{x}_*\mathbf{X}}) \tag{3}$$

One key to good regression accuracy is obviously to use a covariance function (and to a lesser extent a mean function) able to reproduce the observed data well. The strength of the GP methodology lies in the possibility to chose them as members of parametric families of functions – squared exponential kernel for the covariance, linear model for the mean... – and tune their parameters by *maximum likelihood estimation*, that is, by maximizing the marginal log-likelihood (MLL) of the data given the current model parameters. This MLL has the following expression in the case of a Gaussian noise ϵ and a zero mean function $m(\cdot)$:

$$\log p_{\boldsymbol{\Theta}}(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^{\mathbf{T}}\left(\mathbf{K}_\theta + \sigma^2\mathbf{I}\right)^{-1}y - \frac{1}{2}\log|\mathbf{K}_\theta + \sigma^2\mathbf{I}| - \frac{n}{2}\log 2\pi \tag{4}$$

where $\boldsymbol{\Theta}$ denotes model parameters: the noise σ^2 and all parameters θ of the covariance function. This optimization procedure can be performed efficiently by modern automatic differentiation tools and is usually well-behaved (monotonic decrease of the loss function).

2.2 Multi-output Gaussian processes

We now want to perform *multitask regression*, that is, to model simultaneously several correlated quantities – in our case several HXS's. Gaussian processes can be adapted readily to this setup, as long as a correlation model between outputs (regression tasks) is specified [18]. A sensible framework for representing cross-task correlations is to model these outputs as linear combinations of some unobserved and independent GPs:

$$y_j(\mathbf{x}) = m_j(\mathbf{x}) + \sum_{i=1}^{q} H_{j,i}\, u_i(\mathbf{x}) + \epsilon_j(\mathbf{x}) \quad \forall j \in [\![1;p]\!] \tag{5}$$

with p the number of outputs to model, $m_j(\mathbf{x})$ the mean function of the j-th output, u_i's zero-mean and mutually independent Gaussian processes of respective kernels k_i, $\mathbf{H} \in \mathbb{R}^{p\times q}$ the *mixing matrix* which mixes these processes into observed outputs, and the $\epsilon_j \sim \mathcal{N}(0, \sigma_j^2\,\mathbf{I_n})$ task-specific and independent white noise terms[6]. The parameters of the model are: all coefficients of $\mathbf{H}$, the noises σ_j's, all parameters of the kernels k_i's and mean functions m_j's. They can still be tuned by MLL optimization. The underlying Gaussian processes u_i are called *latent processes*, which means that they are a low-dimensional and unobserved embedding of the observations.

The resulting model is dubbed *linear model of coregionalization* (LMC) and is again described in [18]; unfortunately, **the runtime of computing the multi-task versions of Equations (3) and (4) for this LMC is cubic in the number of points n and number of tasks p.** Several approximations or simplifications have therefore been developed in recent years, the most prominent being the *intrinsic coregionalization model* (ICM, [19]) and the *variational LMC*[7] [20]. Another simplification, the Projected LMC, has been introduced by the authors in a yet-unpublished article ([21]). Lastly, a simplified and *training-less* approach, the *Lazy-LMC*, is described in the next section. The description of these models, even with little detail, is cumbersome and outside the scope of this article; we only give an overview of their main features and shortcomings in Table 1, as well as references sufficient to grasp their definition and implementation.

2.3 A convenient, training-less multitask model: the Lazy-LMC

One of the MOGP models used in this work is an *ad hoc* construction absent from the literature, so we describe it here. **The purpose of this model is to be training-less**, that is, to have no free parameter to tune: this way, the interpolation of HXS's can be done directly from the data, without having to resort to an optimization phase and to the associated numerical tools – auto-differentiation library, etc.

[6] Although more complex cross-task noise structures can be specified.

[7] In fact this term doesn't describe a specific model, but rather the application of inducing points and variational approximations to the LMC framework. To our best knowledge no existing paper is dedicated to describing this integration, but clear descriptions can be found in the provided reference.

Table 1. Features of the multi-output Gaussian process models studied in this work.

Model	Advantages	Shortcomings	Refs.
ICM	No approximation.	Computational costs are cubic in the number of tasks – although variants of the model lifting this bottleneck using a low-rank cross-task correlation matrix could be devised.	[19], [25]
	Simple expressions and covariance structure.		
	Closed-form LOO expressions are available (see 4.3).	Lesser expressiveness: all latent processes share the same kernel (same lengthscales, etc.).	
	Accommodates any noise term, thus can be used straightforwardly for uncertainty propagation.		
	Can handle missing data [22] (situation where not all tasks are observed at every input training point).	Requires advanced implementations for large-scale variance computation (see [24]) and other advanced functionalities (inducing points approximations, etc.).	
	The training data can be updated easily (or even with rank-1 updates [23]), without retraining the model.		
VLMC	Computational costs are linear in the number of regression tasks.	"Black-box" model, in which the relationship between the training data and predictions is lost.	[20], [26], [27], [28]
	Very flexible: the latent processes are independent, the loss function (lower bound of the MLL) factors over points and tasks and can be modified in various ways.	In particular, impossible to derive closed-form LOO expressions, or to easily update the training dataset.	
	Natively implements an inducing points approximation, which breaks the cubic complexity in the number of datapoints.	Heavily parametrized (all pseudo-input points are parameters).	
	Accommodates any noise term, thus can be used straightforwardly for uncertainty propagation.	Difficult to ponder the various available approximations, in particular for obtaining well-calibrated uncertainties.	
	Trivially handles settings where the outputs are observed at different input locations.		
PLMC	No approximation, but a constrained noise model.	Convergence of training is slower and/or less stable in some settings.	[21], [24], [29]
	Computational costs are linear in the number of regression tasks.	The constraint put on the noise term prevents usage for uncertainty propagation at the moment.	
	Flexible: the latent processes are independent conditionally upon the training data, the loss function (MLL) factors over points and tasks (except for some cheap terms).		
	Can trivially implement any approximation devised for single-output Gaussian processes, by wrapping the latent processes with it.	Cannot accommodate non-zero mean functions or missing observations at the moment.	
	Closed-form LOO expressions are available (see 4.3).		
	The training data can be updated easily (or even with rank-1 updates [23]), without retraining the model.		
Lazy-LMC	Requires no training.	Non-rigorous model, unable to deliver reliable uncertainty estimations.	2.3, [24]
	Computational costs are linear in the number of regression tasks.	Accuracy not guaranteed if the magnitude of the data noise was misspecified.	
	Closed-form LOO expressions are available (see 4.3).		
	The training data can be updated easily (or even with rank-1 updates [23]), without retraining the model.	Prone to conditioning issues.	
	Can trivially implement any approximation devised for single-output Gaussian processes, by wrapping the latent processes with it.		

Looking at Equation (5) (and the following explanatory paragraph), we can list a number of parameters, which value we have to set without resorting to optimization methods. This job is detailed in Table 2. Several clarifications should be added to this list:

- A zero mean function is retained for each regression task (HXS to model): $m_j(\cdot) = 0 \quad \forall j$. This is a very common assumption in Gaussian process modeling, as the kernel-based component of the model is usually sufficient by itself.
- Setting $\mathbf{H}$ to the principal components of the data matrix $\mathbf{Y}$ is not an arbitrary choice: there is in fact a strong connection between the LMC model and the Probabilistic Principal Component Analysis[8] (see for

[8] In short, the only difference between them is that the LMC assumes a correlated Gaussian structure for each principal component of the data, while the PPCA assumes an uncorrelated one (i.i.d. Gaussians).

Table 2. Description of the Lazy-LMC model. For each parameter featured in the definition of the LMC (Eq. (5)), a procedure is given to set its value to an adequate one. See Section 2.3 for more information.

Term	Description	Procedure to set value or neutralize
$m_j(\cdot)$	Mean function of the j-th regression task	Set to 0
$\epsilon_j \sim \mathcal{N}(0, \sigma_j^2 \mathbf{I_n})$	Noise of the j-th task	Set to a small value (10^{-4}). A noise of this magnitude is also added to the latent processes during interpolation
$\mathbf{H}$	Matrix mixing latent processes into observable signals	Set to the q first left singular vectors, multiplied by their associated singular values, of the data matrix $\mathbf{Y}$
k_i	Kernel of the i-th latent process u_i	Chosen to be the parameter-free *cubic spline kernel* of Equation (6)

instance [30]), so we can hope for this method to yield a good estimate of the optimal mixing matrix.

- It would be delicate to set the parameters of a given kernel to an adequate value without tuning them to the data. Instead, a safer choice is to choose a parameter-free kernel right from the start. To this end, we retained the *cubic spline kernel* of [31], defined in Equation (6), which in 1D emulates a cubic smoothing spline:

$$k(\mathbf{x}, \mathbf{x}') = \prod_{i=1}^{d} \left(1 + u_i v_i + \frac{1}{2} u_i^2 \left(v_i - \frac{u_i}{3}\right)\right) \qquad (6)$$
$$u_i = \min(x_i, x_i')$$
$$v_i = \max(x_i, x_i')$$

  One key feature of this kernel is that it is *non-stationary* (see Sect. 2.4): this guarantees its flexibility, making it able to match the local roughness of the data despite the absence of parameter tuning.

In the end, the philosophy behind this proposed model – dubbed *Lazy-LMC* from now on – is straightforward. To make predictions with it, one simply:

- Computes the *truncated* Singular Values Decomposition (SVD, equivalent to PCA) of the data, for q principal components which will correspond to q latent processes;
- Interpolates these principal components with independent Gaussian processes implementing a parameter-free kernel, as in Equation (6);
- Lifts these interpolations to the original space by applying the inverse PCA transform (multiplication by the matrix of principal directions, i.e in this case the mixing matrix $\mathbf{H}$).

Note however that this procedure is not rigorous, and doesn't guarantee good statistical properties[9]. Its predictions themselves should be accurate, but its measures of uncertainty such as the predictive variance are irremediably flawed.

[9] Indeed, it implicitly assumes that the latent processes are independent conditionally on the observations, which it not the case here. The Lazy-LMC can be framed as a PLMC model ([21]) which main assumption has been overruled rather than enforced.

2.4 Application to the neutronics problem

Let's now specify how this framework can be adapted to the interpolation of HXS's. **A LMC model is here defined at the level of each fuel assembly**. Each row of the training inputs $\mathbf{X} \in \mathbb{R}^{n \times d}$ is a d-dimensional vector describing the state of the considered assembly: in our test case, it is for instance (Bu, T_f, T_m, C_b), with Bu the burnup, T_f and T_m the fuel and moderator temperatures and C_b the boron concentration. Training outputs $\mathbf{Y} \in \mathbb{R}^{n \times p}$ are stacked vectors of p HXS's: each row $\mathbf{Y_i}$ is a complete set of them (for all spatial regions, isotopes, reactions, energy groups...) evaluated at parameter point $\mathbf{X_i}$. Then predictions $\mathbf{y}_*$ at an unknown point $\mathbf{x}_*$ (the interpolation task) are obtained by computing $p(\mathbf{f}(\mathbf{x}_*)|\mathbf{Y})$ as in Equation (3): the mean of this Gaussian distribution is the interpolated values of the HXS's, while its variance represents prediction uncertainty. We further specify below three important choices made in our approach.

Grouping of the outputs. As stated above, we gather all HXS's from an assembly into the same LMC model: this assumes that they can accurately be decomposed as a linear combination of shared "prototypes" (latent processes). This assumption is not hazardous but rather motivated by prior work on the low effective dimensionality of HXS datasets [32]. In fact, even defining one model by assembly is superfluous, since the cited article shows that HXS's are as much correlated across different assemblies as within a given one; we only proceed as such for application-related convenience, and because there would be little computational gain in building an all-assembly model. The reasoning behind this last statement – and more generally, the question of the optimal size of a model for our application – is discussed in Appendix B.

Note that the presented methodology doesn't depart from the standard interpolation pipeline of neutronics

codes: a MOGP model can only be used on the very assembly on which it has been trained (same geometry and materials). More elaborate models could be devised in order to generalize to unseen geometries or material composition, for instance by taking fuel enrichment as a parameter, or by modeling the spatial structure of fuel cells. We leave such possibilities for future work.

Choice of the interpolation points. Unlike traditional methods such as multilinear or spline interpolation, **Gaussian process regression doesn't require a specific structure (multi-dimensional grids) for the interpolation support**: any set of multi-dimensional points can serve as training data. This is a strong argument in their favor as it authorizes more flexibility and above all more sparsity, full grids being notoriously data-inefficient for modeling functions with weak interactions between variables[10] – which is the majority of real-life quantities, HXS's included. However, we must now select a relevant set of interpolation points, with little guidance as to its structure. A principled choice for this can be *low-discrepancy sequences* or space-filling designs, quasi-random sampling methods which fill the domain of interest quickly and evenly. These tools have been thoroughly studied for tasks closely related to interpolation (uncertainty quantification, quadrature...); a popular sequence, Sobol' points, has shown itself to be very effective for these tasks in a large variety of setups [33]. We therefore retained this option in the present work. The good properties of Sobol' points are only guaranteed if the sample size is a power of two, so we stuck to such data sizes in our tests. Here, we only present results for 256 points: we found indeed model accuracy to strongly improve up to this number, then nearly stall[11]. Finally, let us mention that Gaussian process methodology authorizes principled *adaptive sampling* – incremental and intelligent construction of the training set – which we will discuss in the conclusion.

Construction of the covariance function. In Expression (5) of the LMC model, each latent process u_i follows a GP with kernel k_i; we have not provided so far a recipe for these covariance functions. This encompasses at least two questions: the choice of a mathematical function, and this of a dependence structure between variables. For the first one, the most common choice is this of a *stationary* kernel: in this case k_i is a positive function $\psi : \mathbb{R} \mapsto \mathbb{R}_+$ applied to a custom norm between multi-dimensional points, $k_i(\mathbf{x}, \mathbf{x}') = \psi\left(\sqrt{\sum_{j=1}^{d} \left(x_j - x'_j\right)^2 / \lambda_j}\right)$, with λ_j a fitted lengthscale. The choice of a dependence structure between variables refers to kernel composition: a sum or product of kernels being a valid kernel, one can build partial kernels featuring only some of the variables – for instance: $k^{Bu,T_f}(\mathbf{x}, \mathbf{x}') = \psi\left(\sqrt{(x_{Bu} - x'_{Bu})^2/\lambda_{Bu} + (x_{Tf} - x'_{Tf})^2/\lambda_{Tf}}\right)$ – and then combine them appropriately. Note that it is customary to scale the base kernel function k by a learned positive parameter: $k' = \alpha\, k$; this has no effect on the prediction themselves, but directly calibrates the predictive variance, which is linear with respect to α.

In our experiments, we retained for testing four stationary kernels and five groupings of variables, all described in Table 3. The rationale behind our specific choices of groupings was to test several hypotheses, namely, whether it was beneficial to:

- Add an extra kernel containing the burnup variable, so that it can resolve short-scale variations at reactor startup;
- Add another variable (T_f or T_m), which we know to be significantly coupled with the burnup (that is, HXS's are sensitive to the interaction of T_m and Bu), to this extra kernel;
- Have at least one kernel featuring all four variables, or if on the contrary the "Fuel + moderator" decomposition suggested in [34] was sufficient.

Choice of the mean function. We must also specify a choice of mean function for the regression tasks – the m_j's of Equation (5). This choice usually receives much less attention in the GP literature; in fact, the mean function is generally chosen to be zero, to the point that Equation (3) is in many places written without it. This is a sensible choice, because GPs are so flexible that their predictions are little impacted by their mean function, as long as the training data is abundant enough. When it is not set to zero, this function is often modeled as a linear combination of basis functions (linear terms, low-degree polynomials...), as it is done in classical regression models. In this case, it is written as $m(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$, with $\mathbf{w}$ the weights of the linear regression and $\phi(\mathbf{x})$ a vector of basis functions evaluated at $\mathbf{x}$. This regression-like approach can be treated in a bayesian manner along with the GP component: this introduces several complications, discussed in chapter 2.7 of [17]. Otherwise, these weights can be treated as deterministic parameters and tuned by MLL optimization, which is the approach retained here. Note that among our four candidate models (these of Tab. 1), the PLMC is not yet compatible with non-zero mean functions, and the Lazy-LMC will never be as its purpose is to avoid any form of optimization[12] – see Section 2.3. For the other models, four types of mean functions were tested, which are again given in Table 3.

[10] Let us explain why on a simplified case, with two non-interacting variables: suppose we approximate a function $f(x_1, x_2) = f_1(x_1) + f_2(x_2)$ on a cartesian grid, with n points on each axis. Then all points from a given x_1-aligned row only provide information about one value of f_2, as x_2 is constant on this row. One could have taken instead the diagonal points of the grid as a training set, retrieving the exact same amount of information with n points instead of n^2.

[11] In the course of our experiments, we also trained models on other datasets that these used in this article, like Sobol' sequences with different direction numbers, or Latin Hypercube Samples. The results were very consistent with these presented. In fact, chapter 6 of the Ph.D. thesis from which this article is derived ([24]) shows that the obtained accuracy is remarkably insensitive to the layout of the training points.

[12] One could however follow a two-stage approach, first fitting the trend model (for instance a low-degree polynomial) to the data by ordinary least squares, then subtracting it from the signal and fitting the GP model to the residual.

Table 3. Description of all modeling choices tested with a grid search. See Section 2.4 for explanations of the notation $\psi(r)$ for stationary kernels, the lengthscale parameters λ_j, and the notations of variable decomposition. The piecewise-polynomial kernel of order 2, taken from [35], is defined as such : $\psi(r) = (1-r)_+^{j+2}\left(1+(j+2)r+\frac{j^2+4j+3}{3}r^2\right)$, where $j = \lfloor \frac{d}{2}+3 \rfloor$ and $(\cdot)_+ = \max(\cdot, 0)$ denotes the positive part.

Option	Candidates	Candidates description	Parameters
Mean function	Zero	$m(\mathbf{x}) = 0$	None
	Constant	$m(\mathbf{x}) = c$	c
	Linear	$m(\mathbf{x}) = \mathbf{w^T x} + c$	$\mathbf{w}, c$
	Polynomial	$m(\mathbf{x}) = \sum_{i=1}^{3} \mathbf{w_i^T x^i} + c$	$\mathbf{w_i}, c$
Kernel function	Squared exponential	$\psi(r) = e^{-\frac{r^2}{2}}$	λ_i
	Matern 5/2	$\psi(r) = (1+\sqrt{5}r+\frac{5}{3}r^2)e^{-\sqrt{5}r}$	λ_i
	Rational quadratic	$\psi(r) = (1+\frac{r^2}{2\gamma})^{-\gamma}$	γ, λ_i
	Piecewise polynomial	See caption	λ_i
Variable decomposition	None	$k = \alpha k^{Bu,T_f,T_m,C_b}$	α
	Fuel + Mod	$k = \alpha\, k^{Bu,T_f} + \beta\, k^{Bu,T_m,C_b}$	α, β
	Bu + Base	$k = \alpha\, k^{Bu} + \beta\, k^{Bu,T_f,T_m,C_b}$	α, β
	T_f + Base	$k = \alpha\, k^{Bu,T_f} + \beta\, k^{Bu,T_f,T_m,C_b}$	α, β
	T_m + Base	$k = \alpha\, k^{Bu,T_m} + \beta\, k^{Bu,T_f,T_m,C_b}$	α, β

2.5 Effect of the modeling choices

In order to simplify the presentation of the experimental results, we evacuate as of now the question of the impact of the previously-described modeling choices, by showing that all "reasonable" options yield similar results, and subsequently deciding on a common set of them for all our MOGPs. All options depicted in Table 3 were gathered in a *grid search*[13] – a systematical test of all the combinations. They were implemented for single-output GPs, and for our three trainable MOGPs (ICM, VLMC and PLMC – see Sect. 2.2); the performances of the resulting models were measured on the test set described in Section 3.1. The results of this large experiment are tedious to represent, so we will summarize them shortly:

- **"Advanced" modeling choices yield small benefits, in particular for MOGPs**. Choosing a complex mean function or a nonbasic kernel decomposition can only improve accuracy by about 10−15%[14], and can often be useless (for instance with the PLMC) or even detrimental (with the VLMC). This was not unexpected for MOGPs, as the fact that outputs are linear mixtures of latent GPs can make complex model features pointless: for instance, adding variable-specific kernels with shorter lengthscales is less relevant if each output already combines q GPs with various lengthscales. Moreover, their optimization landscape is intricate, meaning that adding complexity to the model may not guarantee convergence to the global optimum.
- **Modeling choices have different effects from one model to another, and from the SOGP case**. For instance, the choice of kernel function has a noticeable impact on the ICM, but not on the other models; complex mean functions hinder the VLMC, but not the other ones. The interactions between these choices also differ for each model.
- **If a specific grouping of variables is used, one sub-kernel should contain all of them.** The "Fuel+Mod" decomposition of Table 3, only option not to meet this criterion, yielded catastrophic results for all models.
- **All standard, long-ranged kernels yield similar results, and this is true for all models**; by this we refer to the *squared exponential*, *Matérn-5/2* and *rational quadratic* kernels of Table 3. Here again, the choice of kernel can only impact predictive accuracy by 10−20% in relative terms, and their ranking varies from one model to another. The main criterion for choice should therefore be computational efficiency; in particular, polynomial truncations of the exponential-based kernels tried here could be put at test.
- **By contrast, the *Piecewise polynomial* kernel yielded significantly worse results in all settings, hindering accuracy by more than 30% in relative terms**. The aim of testing it was to investigate the use of *compactly supported kernels* for greater computational efficiency: it can be seen indeed from its expression in Table 3 that it is zero for $r \geq 1$, meaning that datapoints far away from the predicted one don't contribute to the prediction. This experiment thus advises against local interpolation, as

[13] Which also included 5 values of the number of latent processes q in the range $[15; 40]$ in order to neutralize this factor.
[14] Here, an "accuracy improvement of 10%" doesn't mean that one model is accurate up to 0.1% and the other to 10.1%: it means that the latter is accurate up to 0.11%.

Table 4. Specification of the input domain used in experiments.

Variable	Description	Unit	Minimum value	Maximum value	$N_{\text{points}}^{\text{multilin}}$
Bu	Burnup	MWd/t	75	51 000	29
T_f	Fuel temperature	K	273.15	2073.15	7
T_m	Moderator temperature	K	273.15	600	7
C_b	Boron concentration	ppm	0	2500	3

long-range terms apparently contribute to prediction accuracy; other findings of ours corroborated this idea[15].

All these observations lead to the same conclusion: **simple modeling choices should be favored when using MOGPs for HXS approximation**. More complex options bring little improvement at best and can behave in unsettling ways – which differ from one model to another, one implementation to another, and perhaps one dataset to another. In the remainder of this work, we therefore stick to the following simple choices for all models[16]: **a zero mean function for all tasks; squared exponential kernels for all latent processes; and no specific variable grouping** (each kernel features the four considered variables).

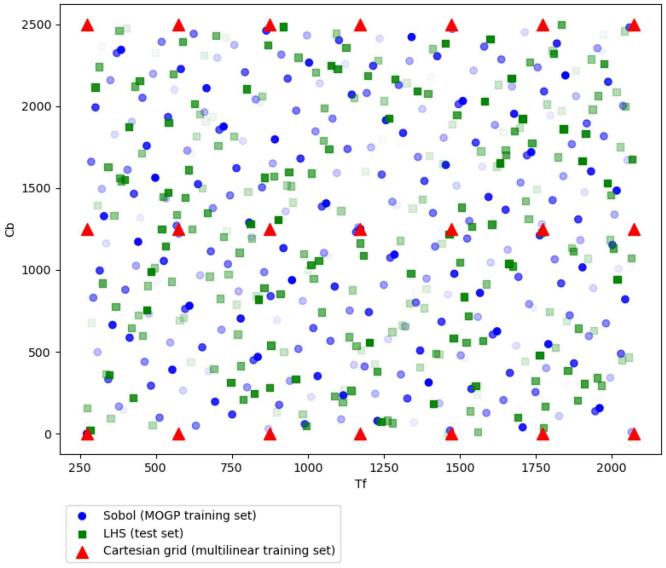


Fig. 2. Schematized view of the three sets of input points used in the experiments, projected in the plane (T_f, Cb). Color fading represents the euclidian distance to the projection plane.

3 Experimental results

3.1 Test-case description

In this talk, we test our models on a full set of *microscopic* HXS's, sufficient to model a standard PWR. The test-case reproduces the notorious VERA benchmark [36]; more specifically, it comprises the twelve assembly types which describe the full core of problem n°5. HXS's matching this benchmark were computed with the neutronics code Apollo3® [37], a deterministic transport neutronic code with multi-channels 1D thermal-hydraulic feedback developed at CEA. They include 40 isotopes, corresponding to actinides, disintegration chains of xenon and samarium, intercycle elements and gadolinium isotopes; only reactions of fission, absorption and scattering were considered. Two discretization schemes were adopted, leading to two increasingly challenging datasets:

- A **standard-discretized dataset**, with 2 energy groups, spatial homogenization at the level of the assembly and 1 anisotropy level for scattering cross-sections: this makes for 287 HXS per assembly (3444 in total), and a training dataset weighing about 3 Mb.
- A **finely-discretized dataset**, with 20 energy groups, spatial homogenization at the level of the fuel cell and 4 anisotropy levels for scattering cross-sections: this makes for about 360 000 non-zero HXS per assembly (4.3 millions in total), and a training dataset weighing about 8 Gb.

As explained in paragraph 2.4, **the retained set of points used for training our MOGPs is a 256-point Sobol' design in 4 dimensions, which value ranges are specified in Table 4**; all variables are defined at the level of the assembly. Burnup values start at 75 MWd/t to match a standard industry practice[17]. Sobol' points having $T_m > T_f$ were also discarded[18] for being non-physical, so that the final training set contains only **201 input points**. The last column of the table describes the grid size used for the multilinear interpolation baseline, to which we compare our method; this grid therefore contains $29 \times 7 \times 7 \times 3 = 4263$ training points. **We emphasize that this grid is only used by the multilinear model, not by the MOGPs**. Finally, **a test set was generated to compare methods on**: it contains 393 points sampled

[15] In particular, forcing the lengthscale parameters λ_i of the squared exponential kernel to small values yielded catastrophic results, indicating that neglecting far-away datapoints was very detrimental to prediction accuracy in our case.

[16] Except for the Lazy-LMC of Section 2.3, for which we explained the necessity of using the spline kernel of Equation (6).

[17] HXS's are considered constant below this threshold in order to keep away from some non-physical behaviors of codes, due to the sudden appearance of some isotopes at reactor start-up.

[18] This doesn't hinder the good properties of the Sobol' design: it just removes a corner of the hypercubic input domain, which will not be used for training or testing.

from a Latin Sampling Hypercube design filling the whole input space (as defined by the value ranges of Tab. 4). More exactly, 512 points were sampled and these with $T_m > T_f$ were discarded as above. We illustrate the three sets of points used in our experiments in Figure 2.

3.2 Experimental specifications

Data postprocessing. Several processing operations can impact the training of the model and/or facilitate the interpretation of error metrics. First, it was noted in [38] that working with $\sqrt{Bu}$ rather than Bu improved interpolation results, as variations are much steeper at low Bu values: we tested and corroborated this claim. Then all input variables were adjusted to the range $[-1;1]$ by an affine transform[19], in order to ease kernel computations and align their characteristic lengths. HXS's were centered and normalized by their maximal absolute value ($\mathbf{y_i} \leftarrow \frac{\sigma_\mathbf{i}-\bar{\sigma}_\mathbf{i}}{\max|\sigma_\mathbf{i}|}$, with σ_i an original HXS), so that each training output y_i has zero mean and values in $[-1;1]$. Centering is crucial as our model has zero mean by construction (we set $\mathbf{m}(\cdot) = 0$), and normalization is too as HXS amplitudes spread over 10 orders of magnitude: a LMC model fitted over un-normalized sections would only focus on these of the largest magnitude. However, this implies storing the original mean and norm of each HXS to renormalize their interpolated values.

Training protocol. Our models contain several parameters: all kernel parameters, the mixing matrix $\mathbf{H}$, task-wise noises σ_i, parameters of the variational approximation for the VLMC... These must be optimized by maximum likelihood estimation: we therefore have to specify a training protocol. Many variants were tried, and optimization proved to be well-behaved in all "reasonable" setups; the only concerns were convergence speed and training budget (allowing enough iterations for the model to converge to the global minimum). We retained the following scheme, well-suited to all of our experiments. The popular Adam optimizer [39] was used for gradient-based optimization; gradients were computed by automatic differentiation with the python `torch` library [40]. Although Adam automatically varies the learning rate of each parameter, we found additional learning rate scheduling to speed up training: an effective policy was to decrease this rate linearly from $\sim 10^{-2}$ to $\sim 10^{-3}$ in a few hundred iterations, then keep it constant. We also implemented a simple stopping criterion, ending training when differences between successive values of the loss (MLL, see Eq. (4)) remained below a relative threshold of $\delta\mathcal{L} = 5 \cdot 10^{-5}$ for a given number of iterations $N_{\text{fluct}} = 50$. Our implementation of GP models is based on the `gpytorch` library [41], and available at the address `https://github.com/QWERTY6191/projected-lmc`. **All experiments were coded in python and performed in double precision, except for this run on GPU with the large dataset, for which single precision was used**. In the course of our experiments, we tried switching from double to simple precision in various contexts; we found no effect on the quality of results, but occasional impacts on the dynamics of model training.

Error metrics. Model accuracy is assessed through the use of an external test set: **we make predictions with the model on the testing dataset described in Section 3.1 (which was not used for training), and compute error metrics on these predictions by comparing them with the true HXS values**. Our retained metrics are the R2 coefficient or *coefficient of determination*, which measures the share of variance in the output that is predictable from the inputs ($R^2 = \frac{1}{p}\sum_{i=1}^{p} 1 - \frac{\sum_{ij}(y_{ij}-\hat{y}_{ij})^2}{\sum_i (y_{ij}-\bar{y}_j)^2}$, with y_{ij} the true value of the normalized HXS j at input location i, $\hat{y}_{ij}$ its predicted value, and $\bar{y}_j$ the mean of the true values of HXS j), the Root Mean Square Error ($\text{RMSE} = \sqrt{\frac{1}{n}\frac{1}{p}\sum_{i=1}^{p}\sum_{j=1}^{n}(y_{ij}^{\text{test}} - y_{ij}^{\text{pred}})^2}$), and the maximal absolute error ($\text{Err}_{\max} = \max_{i,j}|y_{ij}^{\text{test}} - y_{ij}^{\text{pred}}|$). The R2 coefficients should be as close as possible to 1; the RMSE and $\text{Err}_{\max}$ should be comprised in $[0;1]$, as we are working with unit-sized normalized outputs. Notice that **errors computed in this way are expected to be much larger than these reported in the literature**: that's because we work with absolute errors measured on centered and normalized outputs, as opposed to relative errors on raw outputs in the previously-cited sources [3–5,42]. We proceed as such to obtain results consistent with these of the machine learning community: it can indeed be noted that HXS often vary very little around their average value (usually around a few percents and sometimes much lower), yielding relative interpolation errors of 10^{-4}–10^{-5} which are meaningless from a mathematical point of view. In order to produce error metrics closer to the final observables, we also compute errors *at the macro-sections level* – that is, we multiply the absolute error on un-normalized microscopic HXS's by the true concentration of the corresponding isotope at the considered point, to obtain an error in cm^{-1}: $\text{Err}_{\text{mean}}^{\Sigma} = \frac{1}{n}\frac{1}{p}\sum_{i=1}^{p}\sum_{j=1}^{n} C_j^{\text{isotope}(i)}|\sigma_{ij}^{\text{test}} - \sigma_{ij}^{\text{pred}}|$.

We also assess the quality of variance estimation with two metrics: the Predictive Variance Adequacy, $\text{PVA} = \frac{1}{p}\sum_{i=1}^{p}\log\left(\frac{1}{n}\sum_{j=1}^{n}\frac{(y_{ij}^{\text{test}}-y_{ij}^{\text{pred}})^2}{v_{ij}}\right)$, where v_{ij} is the estimated variance at point j and task i, which compares predicted variance with actual errors and should be as close as possible to 0; and α-CI, the proportion of test points falling in the 95% confidence interval, which should therefore be as close as possible to 95%. Finally, we display a storage rate R_{stor}, ratio between the number of floating-point numbers stored by the model and this of the multilinear model (which in this case are only the tabulated HXS values).

Baselines. In order to assess the performances of our GP models for HXS approximation, we wish to compare them with time-tested methods. At the same time, we want to keep easily-implementable methods which require

[19] Except for the spline kernel of Section 2.3, **which requires its inputs to be in** $[0;1]$**.**

little practitioner knowledge. We therefore selected three reference methods, either directly taken from the HXS literature or inspired by it:

- **Multilinear interpolation** (abbreviated as MLI in the following): this is the industry standard (already described in Sect. 1) which we aim to replace, and therefore an obvious baseline to compare to. Of course, this method requires the data to be located on Cartesian grids, contrary to GP models: consequently, the comparison is performed between very different-sized interpolation supports. It can indeed be seen that the grid described in Table 4 features 4263 points, against only 201 for our training Sobol' set.
- **Bayesian Polynomial Regression** (abbreviated as BPR): this is a polynomial regression method with a bayesian treatment of the weights, described in [43]. This model, although not tested in the HXS literature, is similar to existing works with polynomials, in particular [4]; it serves as an intermediate-complexity baseline, more advanced and flexible than MLI (in particular, it is not grid-bound, so it can be compared with MOGPs on the same training data) but less than GPs. Moreover, this bayesian method comes with a variance estimator, which serves as a comparison for this of the GP models. Implementation is done with the `BayesianRidge` class of the `sklearn` library.
- **Single-output GPs** (abbreviated as SOGP): here, we simply use a collection of SOGPs, one for each modeled HXS. These models are trained as in the above paragraph 3.2. Regarding modeling choices, the same grid search as in Section 2.5 was undertaken on these GPs; the retained option was the combination which yielded the best RMSE on average over all HXS's. This baseline serves as a best-estimate model: we wish to assert whether using low-dimensional, multi-output models such as MOGPs incurs a loss of accuracy compared to a collection of GPs, each fine-tuned to a specific HXS.

One advantage of these baselines is that they have few hyperparameters and thus require little hand-tuning; MLI, for starters, has none. BPR has four hyperparameters, which relate to priors on the weights of the regression: the original article describing the method [43] suggests setting them to 10^{-6} to make the prior non-informative, which we did. We also had to provide a maximal degree for the polynomial features: we selected a value $D_{\max} = 7$, as the accuracy of the model didn't improve beyond this value. Lastly, a maximal number of iterations and a tolerance value must be provided for the fitting of the model; these options were found to be non-impactful as long as they were selected large enough and small enough respectively.

3.3 Effect of the latent dimension

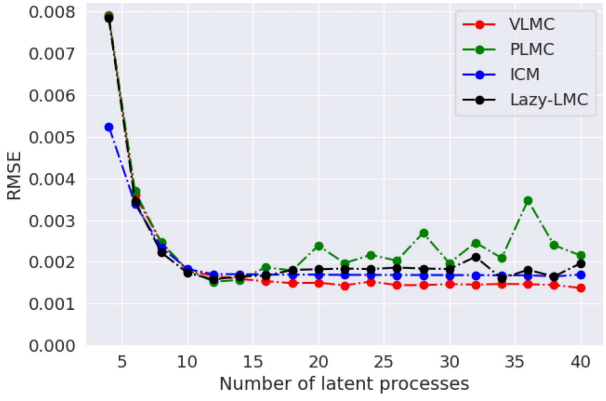
Before comparing our MOGPs to the baselines, we must make an important choice: this of a latent dimension, i.e of a number of latent processes (see Eq. (5)). Optimizing it is key for performance, as we will see in Section 4: both the computational cost of predictions and the storage requirements of the model are linear in this parameter q. On the other hand, its value must be large enough for the low-dimensional space to accurately represent the data.

We therefore wish to study how this parameter affects the accuracy of predictions. To this end, **for each even value of q ranging from 4 to 40, we train and fit an instance of each MOGP with q latent processes on one assembly of the standard-discretized dataset, make predictions on the external test-set, and compute the RMSE between these predictions and the true HXS values**. The results are shown in Figure 3a. They are very clear: **for all models, the RMSE decreases sharply until $q = 12$, then stalls**. This of the PLMC even increases beyond this point, showing signs of instability. The perfect agreement of the 5 models between $q = 6$ and $q = 18$ is remarkable, and very reassuring, as it tends to indicate that they are all fitted to their best[20]. In the same vein, it can be observed that **the RMSE values displayed on the graph are excellent**: the data being normalized to unit size, a RMSE of 0.002 is a synonym of relative errors around 0.2% on average.

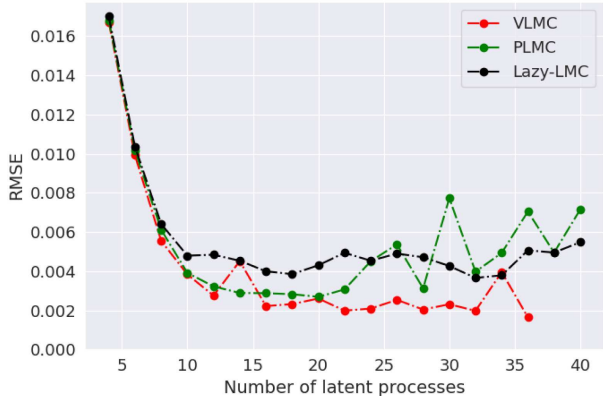
The stalling of the RMSE above some value of q is surprising: our models seem to be unable to exploit the additional information brought by more latent processes. One possible explanation is that the optimization procedure is not guaranteed to find the global optimum of the MLL for large values of q: an intricate optimization landscape could prevent training from resolving the fine-grain details of low-amplitude latent processes. Another one is that the information gain brought by additional latent processes may be of the same magnitude as the interpolation error made on them: in this case, it could act as "noise" at prediction time and yield no accuracy improvement. In order to test this last hypothesis, we can simply perform truncated SVDs of the considered training data matrix $\mathbf{Y}$ with increasing values of the truncation rank r, and plot the resulting compression losses against r. We define this compression loss as $1 - \mathrm{EVR}(r)$, with $\mathrm{EVR}(r)$ the *cumulative explained variance ratio* of the SVD – the sum of the r computed singular values divided by $||\mathbf{Y}||_F$, such that $\mathrm{EVR}(p) = 1$. The results are shown in Figure 4. They are in line with our hypothesis: **for $r = 12$, only 10^{-4} of the data variability is lost to SVD truncation, which is smaller than the accuracy of our SOGP interpolants measured in Table 5**. It is therefore plausible that adding latent processes beyond this value only brings "noise" to the model, whether it is at training time or at prediction time. It is important to notice that curves Figures 4a and 4b are almost identical: **the finely-discretized dataset contains scarcely more information than the standard-discretized one**, as in both cases this information is almost entirely contained in the first 30 principal components.

We can also inquire on the optimal value of q for the finely-discretized dataset, containing $p \simeq 3.5 \cdot 10^5$ HXS's per assembly. We thus perform the same experi-

[20] The VLMC and PLMC are both very good approximations of the underlying LMC model, so we expect their results to be nearly identical if they are well-optimized.



(a)



(b)

Fig. 3. Variation with the number of latent processes q of the prediction error (RMSE measured on one assembly of the test dataset) for the studied MOGP models. (a) Standard-discretized dataset. (b) Finely-discretized dataset.

ment as in Figure 3a on this dataset; results are shown in Figure 3b. Notice that the ICM is absent from it: **in available implementations of this model, computational complexities contain a term $O(p^3)$, making it intractable for large numbers of HXS's**[21]. The main observation is that **we still observe a plateau in the progress of the RMSE with q, starting at about the same value**. With the PLMC, the same signs of instability as for the smaller dataset (Fig. 3a) are visible. Another interesting observation is that **this time, the trainingless model Lazy-LMC performs significantly worse than its competitors**: its best RMSE is twice larger than the best value for the VLMC.

Returning to our initial question of the optimal choice of q balancing accuracy with computational performance, one important conclusion stands out: **even the largest HXS datasets are of extremely small effective linear dimension, which makes it possible to use low-dimensional approximations on them with as few as a dozen of latent components, at the cost of an accuracy loss which is indistinguishable from interpolation error**. We therefore retain $q = 16$ in the next experiments.

3.4 Comparison with the baselines

We start by testing our methodology on the standard-discretized dataset described in Section 3.1, performing a simple comparison between our regressors: **we train a multi-output model (or a collection of models for the baselines) for each of the 12 fuel assemblies, make predictions with them on the external test set, compute error metrics and aggregate them**. Results are displayed in Table 5; the training time for each assembly-wise LMC model was about 2 minutes on a standard CPU machine. Note that variational GPs, of which our VLMC model is an example, make use of *inducing points*, pseudo-observations meant to "replace" the more numerous original ones: we retained a number $n' = \lfloor n/1.5 \rfloor$ of them, with locations identical for all latent processes and learned by the model[22]. This choice was once again made for optimal accuracy: putting $n' = n$ fixed inducing points at the locations of the observed data would theoretically be more accurate, but proved less stable in practice. A few main observations can be made from Table 5:

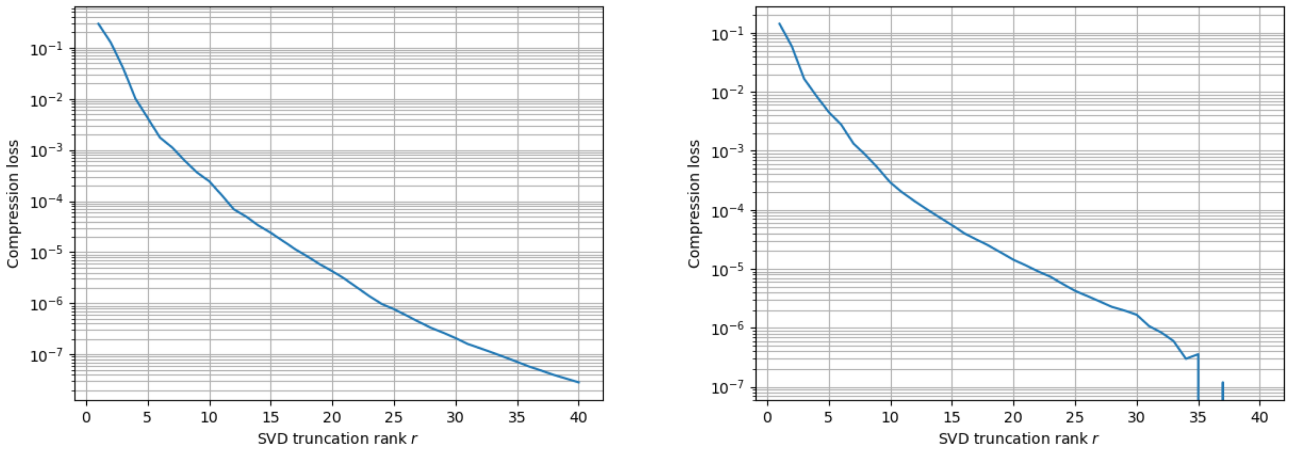
- **All four MOGP models have very similar performances**, as already noticed in Figure 3a.
- **Their accuracy is overall excellent**: all $R2$ coefficients are very close to 1 in particular. All error metrics are very close to these of the collection of SOGPs, even though major dimensionality reduction was achieved, and simpler modeling choices retained for MOGPs (squared-exponential kernel, no complex mean or variable decomposition). MOGPs also best the BPR baseline in terms of RMSE and maximal error.
- **GP models reach better accuracy than the multilinear baseline with 20 times less interpolation points**[23].
- **Training a MOGP is much faster than training a set of SOGPs** – about 500 s in the first case versus 16 600 s in the latter. These SOGPs were however trained sequentially rather than in parallel: if we assume that 40 such models can be trained simultaneously on a single machine with the same training time per model as in the sequential setting[24], the training duration of the SOGPs would be reduced to 415 s. Although it is not displayed in Table 5, **making predictions with MOGPs is also much faster than making them with a set of SOGPs**, as it will be explained in Section 4.1.

[21] An alternative implementation lifting this shortcoming is however suggested – but not tested – in the PhD thesis from which this article is derived ([24]).

[22] This is also true for Section 3.3.

[23] The only metric which is more favorable to multilinear is $\mathrm{Err}^{\Sigma}_{\max}$, the maximal error after multiplication by concentrations. Even here the difference is slight, and could be mitigated by better support selection or physics-informed normalization.

[24] Which is unlikely, as the linear algebra routines performing the sequential training had access to 40 processors.



(a) (b)

Fig. 4. Compression loss $1 - \mathrm{EVR}(r)$ of the training data (normalized HXS's of one assembly) in function of the truncation rank r of the SVD, where $\mathrm{EVR}(r)$ is its *cumulative explained variance ratio*. The erratic behavior of the left part of subfigure b/ is due to the compression loss reaching single-float precision (while figure a/ was computed in double-float precision). (a) Standard-discretized dataset. (b) Finely-discretized dataset.

Table 5. Performance metrics on all assemblies of the standard-discretized dataset, for the studied MOGP models (all with $q = 16$ latent processes) and three baselines. The best and worst value for each metric are highlighted in green and red respectively. Definitions of the metrics are given in Section 3.2; they are all averaged across all HXS's. The Err^{Σ} metrics were not computed for the polynomial method, which was tested at a time when concentration data was unavailable. The training time T_{train} is defined at the level of one assembly. n is the number of datapoints used for training.

Model	$R2$	RMSE	$\mathrm{Err}_{\mathrm{max}}$	$\mathrm{Err}^{\Sigma}_{\mathrm{mean}}$ (cm^{-1})	$\mathrm{Err}^{\Sigma}_{\mathrm{max}}$ (cm^{-1})	α-CI	PVA	T_{train} (s)	$\mathcal{R}$	n
ICM	0.997	$2.01 \cdot 10^{-3}$	$6.14 \cdot 10^{-2}$	$2.37 \cdot 10^{-4}$	0.363	0.824	−2.15	354	142	201
VLMC	0.994	$1.61 \cdot 10^{-3}$	$5.10 \cdot 10^{-2}$	$1.71 \cdot 10^{-4}$	0.957	1.00	−5.92	260	167	201
PLMC	0.995	$1.56 \cdot 10^{-3}$	$2.62 \cdot 10^{-2}$	$1.52 \cdot 10^{-4}$	0.211	1.00	−7.20	668	141	201
Lazy-LMC	0.999	$1.34 \cdot 10^{-3}$	$3.23 \cdot 10^{-2}$	$8.13 \cdot 10^{-5}$	0.173	1.00	−11.6	$5 \cdot 10^{-4}$	142	201
MLI	0.998	$2.20 \cdot 10^{-3}$	$4.47 \cdot 10^{-2}$	$1.69 \cdot 10^{-4}$	$9.57 \cdot 10^{-2}$	–	–	0.806	1	4293
BPR	0.998	$2.80 \cdot 10^{-3}$	0.102	Not computed	Not computed	0.982	−0.792	3.90	13	201
Best SOGP	0.999	$1.36 \cdot 10^{-3}$	$3.03 \cdot 10^{-2}$	$3.75 \cdot 10^{-5}$	$4.81 \cdot 10^{-2}$	0.945	−0.625	16 600	19	201

- **MOGPs offer major storage savings**: a value $\mathcal{R} = 167$ means that the model occupies 167 times less memory than the lookup table required for multilinear interpolation. This is one order of magnitude better than the gains achieved with SOGPs or BPR; this performance is made possible by dimensionality reduction.
- **The performance of Lazy-LMC is surprising**: its accuracy rivalizes with or even surpasses this of other MOGPs, while it is the only one of them which doesn't require training – hence its tiny T_{train} metric, which only corresponds to fitting duration in this case. This makes it a very attractive applicative choice if simplicity is desired.
- Lastly, **the predictive variance of the MOGPs appears to be very ill-calibrated**: a PVA of −5.87, for instance, means that predicted uncertainties are on average $\sqrt{\exp(5.87)} \simeq 19$ times larger than actual errors. In these conditions, all test points fall inside the 95% confidence interval, hence the α-CI values of 1.00 for VLMC, PLMC and Lazy-LMC. For ICM, the situation is more intricate: this model has at the same time a very negative PVA, indicating a significant overall overestimation of uncertainties, and an α-CI well below 0.95, showing that uncertainty is on the contrary underestimated for a significant fraction of all points. We explore the behavior of these variance estimates in the next subsection, to hint whether there is still hope for UQ of neutronics data with MOGPs.

In short, **apart from variance estimation, MOGPs are up to their promise: they deliver very accurate predictions – better than multilinear interpolation and polynomial regression, almost as good as the best-performing SOGPs – with a very low training cost, and allow very fast predictions and major storage savings**. For the moment, all four of our MOGP models are still in the running, as they all have very similar predictive performances; the absence of training of the Lazy-LMC, however, is a significant advantage,

which authorizes in particular extremely simple implementations (no autodifferentiation library is needed).

Results for the finely-discretized dataset are very similar to these presented in Table 5, with some subtleties to be discussed; we therefore defer it to Appendix A. We only emphasize that **three of our MOGP models (VLMC, PLMC and Lazy-LMC) could be successfully trained and/or fitted on $\simeq 3.5 \cdot 10^5$ HXS's, yielding predictive RMSEs of the order of 10^{-3}; for the PLMC, training was completed in only 2 minutes on a GPU**. This proves that this methodology is fully scalable to even the largest current HXS datasets.

3.5 Improving the predictive variance

In the previous section, we found MOGPs to yield ill-calibrated variance estimates. This deserves investigation: although uncertainty quantification is not a primary goal of this work, it is nevertheless a growing subject in neutronics, in which GPs could open new avenues. After exploring several leads, **we found three main factors impacting the predictive variance, which can become very well-calibrated if the right choices are made**.

Fixing conditioning issues. It is a well-known problem in GP literature that **kernel matrices tend to be poorly conditioned** [44,45]. This has a natural explanation: for two nearby training points $\mathbf{x_i}$ and $\mathbf{x_j}$, the corresponding rows $k(\mathbf{x_i}, \mathbf{X})$ and $k(\mathbf{x_j}, \mathbf{X})$ in the kernel matrix $\mathbf{K}$ are very similar if the kernel function is smooth, meaning that $\mathbf{K}$ is close to singular. The noise parameter σ^2 (see Eq. (3)) acts as a regularization term in the inversion of $\mathbf{K}$, as it adds a positive diagonal term to it; noise being closely connected to variance estimation, one can suspect that wrong predictive variances are linked to conditioning problems. We found indeed the condition numbers of kernel matrices to be very large in our case – up to 10^{10}.

One way in which conditioning can impede the predictive variance is through an eventual *noise lower bound*. Indeed, it is sensible for a GP implementation to have a noise thresholding mechanism, in order to keep all computations well-conditioned during model training; this is the case in the `gpytorch` library which we use. **In our case, the modeled data is so smooth that we observe fitted values of σ to match this lower bound σ_B in all situations**. We can thus suspect that these fitted noises are constrained to artificially high values, and that better variance estimates could be obtained by decreasing the lower bound and letting noise terms reach their "true" ranges. Experiments confirm this intuition: **when we decrease σ_B, we see all computation times to increase sharply due to worse conditioning, but the predictive variance to become much better-calibrated**. There is therefore a trade-off to make between variance estimation and speed. One could also try to improve the conditioning of kernel matrices by other means, in order to be able to lower noise values without sacrificing computation times – for instance by pruning the training data as suggested in Section 4.3, or by resorting to the preconditioning techniques and sparse approximations of kernel matrices presented in [45]. One such improvement is available on-the-shelf for the VLMC: it suffices to decrease the number of *inducing points* which the variational approximation relies on.

Moving to standard normalization. As stated in Section 3.2, we processed each HXS by normalizing it to its maximal value. The reason for this choice was to work wit unit-sized quantities, so that absolute errors could be easily interpreted; this way we could avoid the computation of relative errors, where problems of divisions by zero occurred. However, for any vector $\mathbf{f}$ there stands $\max(|\mathbf{f}|) \geq \mathrm{std}(\mathbf{f})$, so that HXS's normalized with this scheme can have any variance between 0 and 1; it happens that for some of them $\mathrm{var}(\mathbf{f}) \ll 1$. We can make the hypothesis that gathering tasks with disparate variances inside a single LMC model is troublesome for its variance estimation; we therefore tried to normalize the data by its standard deviation instead, and to assess the predictive variance of the models in this setting. Once again, **a drastic improvement of the predictive variance is observed: we thus conclude that the training data for each task of the MOGP should have variance $\simeq 1$ for the predictive variance of the model to function well**. It is interesting to note in Table 6 that this change of normalization also improves on the model's R2; this suggests that the outputs of the smallest variance are correctly addressed in this setting, whereas they were neglected by the model in the previous one[25].

Making noise-appropriate modeling choices. One last possible cause of poor variance estimation is straightforward: any GP model makes assumptions about the noise in the data, and these assumptions must be appropriate for uncertainty estimation to work well. A discussion of these modeling choices is way beyond the scope of this article, at it delves deep into the mathematics of the studied models. Let us just mention the options which yielded the best results, and refer to the literature for an explanation of them:

- For the VLMC, the important factors are the chosen *variational distribution* and *loss function* – which in this model is not the standard MLL (Eq. (4)), but a lower bound of it. The variational distribution refers to the probability density which is used to parametrize the posterior distribution of the model; we found the most effective choice to be the standard one of using a fully-parametrized multivariate Gaussian, instead of simplified options such as the Dirac distribution ("delta variational distribution") which we used in previous experiments. Regarding the loss function, several options are available in the literature, the most popular today probably being the ELBO of [26]; but some

[25] Contrary to all of our other metrics, the R2 coefficient relates the error measured on each regression task to the variance of this task, so it is the only one to detect if the model is "lazy" on low-variance ones (it makes low prediction errors, but which are in fact significant compared to this task's variability).

Table 6. Summary of the process of fixing predictive variance for VLMC and PLMC. Each row denotes a modification performed on them and the corresponding results. These modifications are incremental: each one is kept in the following rows.

	VLMC			PLMC		
Modification	$R2$	PVA	α-CI	RMSE	PVA	α-CI
Baseline	0.994	−5.60	1.00	0.993	−7.20	1.00
Decreasing σ_B^2 from 10^{-4} to 10^{-7}	0.998	−0.547	0.88	0.999	−7.69	1.00
Standard normalization (rather than normalization by the maximal absolute value)	0.999	3.12	0.388	0.999	−1.30	0.996
Model-specific options favorable to variance estimation (Gaussian variational distribution and "predictive likelihood" loss function for VLMC, general noise model for PLMC)	0.999	0.484	0.893	0.999	−1.02	0.995

have been designed which specifically aim at improving the predictive variance, for instance the so-called "Predictive Log-likelihood" [28]. We found indeed that training the model with this PLL rather than the standard ELBO improved its variance estimation, while preserving its accuracy.

- The PLMC, in its turn, is based upon a restriction of the general noise model of the LMC [21]. This restriction can be made more or less drastic, the harshest choices yielding the best computational performance for the model; yet one can assume that the less-restricted noise models will give the most faithful variance estimation. This is indeed what we observed: the least-simplified variants of the PLMC should therefore be used if the quality of variance estimation is favored over speed.

The above experiments are summarized in Table 6, in a cumulative fashion: each change made to the model is kept in the following rows. It can be seen that **the added effects of the suggested changes turn the calibration of predictive variance from catastrophic to very decent**: the VLMC exhibits a PVA of 0.484, meaning that the actual errors are on average only $\sqrt{\exp(0.484)} \simeq 1.27$ larger than the predicted uncertainties; likewise for the PLMC, $\sqrt{\exp(-1.01)} \simeq 0.60$. The α-CI values for these models mean that respectively 89.3% and 99.5% of prediction errors fall within the 95% predicted confidence interval, close to the desired value (95%). Further work on conditioning and modeling choices could still improve on these results.

Case of the ICM and Lazy-LMC. We weren't able to achieve such good results with the ICM. The change in normalization yielded somewhat acceptable variance estimation, with PVA $= -0.70$ and α-CI $= 0.80$. Notice however that these two values point towards opposed conclusions: PVA < 0 suggests overall overestimation of errors, while α-CI < 0.95 indicates the opposite. This means that errors are underestimated for many HXS's, and very overestimated for others: there is not only a problem of calibration, but also of estimation quality. It is unclear how to improve on this situation; there are hints that this problem may stem from linear algebra issues, as the conditioning problem remains vivid, and as we will see in Section 4.3 that another kind of computations with this implementation of the ICM exhibits inconsistencies. Efficient computations with this model indeed rely on subtle Kronecker-product-based operations and eigenvalue computations, as described in [25], making it sensitive to implement details.

Lastly, no modification had a positive impact on the predictive variance of the Lazy-LMC, which PVA always remained below the catastrophic value of −11. Several reasons can explain this underperformance: the training-less SOGP modeling each latent process already has very poor variance estimates, probably because it doesn't feature a *variance scaling parameter* (see Sect. 2.4), and perhaps also because of its peculiar *spline kernel*, which is not even derivative-continuous. The covariance structure of the latent processes of the Lazy-LMC is also inconsistent, as explained in Section 2.3. There is therefore little hope that this model can ever yield decent uncertainty estimates.

4 Practical considerations

4.1 Computational performance of interpolation

We didn't include measurements of prediction durations in our experimental results of Section 3.4. This is because such values would be heavily implementation-dependent; the methods tested here being only prototypes written in Python with little consideration of performance, they would be little representative of industry-grade implementations. However, we already mentioned interpolation speed to be critical in neutronics applications; assessing this performance in some manner is thus mandatory.

We therefore propose here a *theoretical* analysis of the computational cost of MOGP prediction. We focus on the computation of the predictive mean (the interpolated values), assuming that prediction speed is much less critical for variance estimation and UQ-related tasks. In the following paragraphs, we compare how predictions are made with SOGPs, MOGPs and multilinear interpolation, in order to compute their costs and identify their respective sources of over- or under-performance.

Cost of predictions with multilinear interpolation. We start with predictions of the MLI model, as the reasoning behind this analysis will help to grasp the issues posed by the other models. A prediction with MLI at the input point $\mathbf{x}_*$ is represented by the following operation:

$$\hat{y}(\mathbf{x}_*) = \sum_{i=1}^{2^d} \alpha_i(\mathbf{x}_*)\, y_i^* \tag{7}$$

where the y_i^*'s are the values of the HXS at the 2^d grid points which surround the test location $\mathbf{x}_*$. The coefficients $\alpha_i(\mathbf{x}_*)$ – which expressions are given in [46] for $d = 2$[26] – are geometrical in nature: they only depend on the coordinates of $\mathbf{x}_*$, **not on values of y. They are therefore identical for all HXS's**. Consequently, as long as the number of modeled HXS's is large enough, that is, $p \gg 2^d$, the cost of computing the α_i's is negligible compared to this of performing the p linear combinations of Equation (7). The total cost of making predictions (defined as the number of required floating-point operations) is then:

$$\mathcal{C}_{\text{MLI}} \simeq 2 \times 2^d p \tag{8}$$

Cost of predictions with SOGPs. Gaussian processes being *linear estimators* – predictions are linear combinations of the known values y_i – the cost of making predictions with them follows the same logic as for MLI. A prediction with a zero-mean SOGP which can indeed be written as:

$$\hat{y}(\mathbf{x}_*) = \mathbf{k_*}^T \mathbf{K}^{-1} \mathbf{y} = \sum_{i=1}^{n} \left[\mathbf{k_*}^T \mathbf{K}^{-1}\right]_i y_i = \sum_{i=1}^{n} \alpha_i^y(\mathbf{x}_*)\, y_i \tag{9}$$

$$= \sum_{i=1}^{n} \left[\mathbf{K}^{-1}\mathbf{y}\right]_i k(\mathbf{x}_*, \mathbf{x_i}) = \sum_{i=1}^{n} C_i\, k(\mathbf{x}_*, \mathbf{x_i}) \tag{10}$$

where we recall that $\mathbf{x_i}$ is the i-th training point, $\mathbf{K} = k(\mathbf{X}, \mathbf{X})$, and $[\cdot]_i$ denotes the i-th component of a vector. The first line of Equation (9) aims at highlighting the similarity with the multilinear case: in this case too, the prediction is a linear combination of the observations y_i, which coefficients $\alpha_i^y(\mathbf{x}_*)$ are geometrical functions[27] of the test location $\mathbf{x}_*$. There are however two important differences: this linear combination now has size n instead of 2^d, and the coefficients α_i^y are different for each HXS, as a distinct kernel has been trained for each.

The second line of Equation (9) highlights a different aspect of this computation: the product $\mathbf{K}^{-1}\mathbf{y}$ is independent on $\mathbf{x}_*$, so **it can be precomputed in an offline phase to make a cache C**. The cost of making predictions for an HXS is then the cost of computing the n kernel values $k(\mathbf{x}_*, \mathbf{x_i})$, plus this is the linear combination of size n. If we note c_k the cost of a single kernel evaluation, the total cost of making predictions for p HXS's is then:

$$\mathcal{C}_{\text{SOGP}} = n(c_k + 2)p \tag{11}$$

This compares unfavorably to MLI, for which $\mathcal{C}_{\text{MLI}} \simeq 2 \times 2^d p$. For one thing, $(c_k + 2)$ is expected to be much greater than 2, as the cost of a kernel evaluation is generally much higher than this of a simple multiplication[28], at least by one order of magnitude. Moreover, n is expected to be significantly larger than 2^d – all training points are involved in the computation, instead of just the nearest ones: in our case for instance, $n = 201$ while $2^d = 16$. The lesson here is clear: **even when precomputing all quantities that can be, predictions with SOGPs involve two costly operations: the computation of a kernel value at each of the n training points, and the linear combination of these n values with precomputed coefficients**. This applies to any long-ranged, kernel-based regressor; such methods will always have prediction costs at least two orders of magnitude larger than MLI. Fortunately, **MOGPs can bypass this bottleneck, as they don't perform kernel operations for each HXS, but only on a much smaller collection of latent processes**.

Cost of predictions with multi-output GPs. Predictions with the VLMC, PLMC and Lazy-LMC[29] are performed in two steps: first, predictions are made for the independent latent processes of the MOGP, as in Equation (9); then these latent values are mixed by a multiplication with the mixing matrix $\mathbf{H}$ (see Eq. (5)).

$$\hat{u}_j(\mathbf{x}_*) = \mathbf{k_{j*}}^T \mathbf{K}_j^{-1} \mathbf{U_j} = \sum_{i=1}^{n} C_{ij}\, k_j(\mathbf{x}_*, \mathbf{x_i}) \quad \forall j \in [1; q] \tag{12}$$

$$\hat{y}_l(\mathbf{x}_*) = \sum_{j=1}^{q} H_{lj}\, \hat{u}_j(\mathbf{x}_*) \quad \forall l \in [1; p], \tag{13}$$

where we recall from Section 2.2 that the u_j's are unobserved latent processes of respective kernel k_j. Here, $\mathbf{U_j}$ denotes a low-dimensional summary of the training data "feeding" the j-th latent process; the expression of this summary differs from one of our models to another, so we refer to the fifth chapter of [24] for a complete exposition. The important point is that **it can be precomputed in an offline phase, meaning that the term $\mathbf{C_{ij}}$ in Equation (12) is a precomputed cache**.

The cost of predictions with such models can be read from Equation (12). The first line is the computation of q SOGP predictions, which represents $n(c_k+2)q$ operations according to the previous paragraph, while the second line is simply a matrix-vector product of size $pq \times q$; the total cost is then:

[26] For higher dimensions, they are rather obtained in a recursive manner, as in [1].

[27] In this case, the geometry is this defined by the kernel.

[28] One way around this issue would be to mutualize kernel operations between all SOGPs of the collection, either by enforcing them to share the same kernel parameters, or by resorting to a parameter-less kernel like this of Equation (6). The prediction cost would then go down to $2np + nc_k$.

[29] The case of the ICM is more complex; we refer to the fifth chapter of [24] for an explanation, and a comparison to Equation (12).

$$\mathcal{C}_{\text{LMC}} = n(c_k + 2)q + 2pq \tag{14}$$

The key point here is that **the costly GP interpolation now only applies to a small number $q \ll p$ of latent processes**. Let us consider the case of large HXS datasets, that is, $p \gg n(c_k + 2)$; in this case $\mathcal{C}_{\text{LMC}} \simeq 2pq$. Remind that $\mathcal{C}_{\text{MLI}} \simeq 2^{d+1}p$, so that comparing the interpolation speed of MLI and LMC in this setting amounts to compare 2^d and the number of latent processes q. We saw in Section 3.3 that $q \simeq 10$ is optimal even for our finely-discretized dataset; one can expect similar values for other test-cases[30]. Therefore, **for large HXS datasets, the cost of making predictions with the LMC is identical to or even better than this of MLI**. This performance is made possible by dimensionality reduction: **in this regime, the main cost is not the interpolation task, which is confined to the low-dimensional space, but the lifting of the low-dimensional interpolated values to the original space**.

However, this many-HXS regime may not apply to all situations encountered in industry calculations: for instance, in our representative standard-discretized dataset with $p = 287$ HXS per assembly, assuming $n \simeq 200$ and $c_k \simeq 30$ (piecewise-polynomial or spline kernel), there stands $n(c_k + 2) \simeq 6000 \gg 287$. Prediction costs are then one order of magnitude higher than these of MLI. There are however two ways to alleviate this burden:

- **Reducing the number of training data n**: the training support can be pruned by *downsampling* (see Sect. 4.3) or built incrementally from scratch by *active learning* (Sect. 5). Preliminary experiments in [24] show that values as low as $n = 100$ can be reached on our test-case without loss of accuracy. Besides, for the VLMC, it is in fact not the training points, but fewer virtual *inducing points* which are used at prediction time; here again, numbers $m < 50$ of inducing points were tested in [24] with preserved accuracy.
- **Mutualizing kernel computations**: if the same kernel is used across all latent processes, the number of kernel operations goes from $n(c_k + 2)q$ to $n(c_k + 2)$. This is the case if the spline kernel of Equation (6) is used for instance, as it has no parameter which can differ from one latent process to another. **In such a case, the computational cost falls back to the order of this of MLI.**

Also mind that this analysis only addresses arithmetic operations, not memory transfers. The operations described here being elementary (small matrix-vector products) and the volume of involved data being large, **HXS interpolation is in fact likely to be cache-bound or memory-bound in many settings and implementations. In this case MOGPs would have the edge over MLI, as they work with much smaller data**, as we will see it in the next section.

[30] An argument in this direction is the fact that the main source of HXS variability is their homogenization flux, which is common to all HXS's in a given assembly and energy group.

4.2 Model storage and interpretability

One of the objectives of the present work is to reduce the size of HXS data libraries, which can get very large today for best-estimate simulations (up to hundreds of gigabytes). MOGPs achieve this goal in two manners: by reducing the number of training samples to store for interpolation, and by leveraging dimensionality reduction. It can be seen in Tables 5 and A.1 that the volume of HXS libraries can be divided by a factor of more than 200 this way. The only quantities to store, which completely define the model, are: the mixing matrix $\mathbf{H} \in \mathbb{R}^{p\times q}$, the matrix $\mathbf{U} \in \mathbb{R}^{q\times n}$ collecting the quantities $\mathbf{U_j}$ introduced in the previous section (low-dimensional summaries the data "seen" by the latent processes), the matrix of training data inputs[31] $\mathbf{X} \in \mathbb{R}^{d\times n}$, and the few parameters of the model (task-wise noises and kernel parameters for the q latent processes). Neglecting the latter[32], the number of floating numbers to store is thus $pq + qn + nd$, to be compared to the np values contained in the matrix of training observations $\mathbf{Y}$.

It is interesting to note that **for the PLMC and Lazy-LMC, the form in which the model is stored matches a very common matrix compression framework**. For these models, $\mathbf{U}$ has a simple expression given in [21]: $\mathbf{U} = \mathbf{TY}$, with $\mathbf{T}$ some generalized pseudoinverse of the mixing matrix $\mathbf{H}$ (i.e, $\mathbf{TH} = \mathbf{I_q}$). This means that $\mathbf{HT}$ is a projection matrix ($(\mathbf{HT})^\mathbf{2} = \mathbf{HT}$), and therefore it stands:

$$\mathbf{Y} \simeq \mathbf{HTY} = \mathbf{HU} \tag{15}$$

In other words, **multiplying together the two stored matrices defining the model – $\mathbf{H}$ and $\mathbf{U}$ – yields a rank-q projection of the original data $\mathbf{Y}$**. Such a factorization is a common manner of compressing a matrix of low effective rank; for instance, SVD-based compression is a staple in many fields. Incidentally, recall from Section 2.3 that the definition of the Lazy-LMC involves Singular Values Decomposition: if $\mathbf{U\Sigma V^T}$ is the ranq-q truncated SVD of $\mathbf{Y}$, $\mathbf{H} = \mathbf{U\Sigma}$ and $\mathbf{U} = \mathbf{V^T}$. The property $\mathbf{Y} \simeq \mathbf{HU}$ is beneficial in terms of model interpretation: one can dispense with storing the full (potentially large) data matrix $\mathbf{Y}$, but still recover an optimal rank-q projection of it by a simple multiplication of the stored $\mathbf{H}$ and $\mathbf{U}$.

Also note that this format makes for efficient computations. $\mathbf{U} \in \mathbb{R}^{q\times n}$ being very small, it can be stored in low-level caches during predictions; moreover, $\mathbf{X}$, $\mathbf{H}$ and $\mathbf{U}$ can each be stored contiguously and are fully read during any interpolation work. By contrast, with multilinear interpolation, the 2^d grid points nearest to the prediction location must be searched for each single HXS inside the large tensor of its tabulated values, in a discontiguous manner.

[31] For the VLMC, training datapoints are replaced by the less numerous *inducing points*.

[32] A complete counting of stored quantities is given in the appendices of [24].

4.3 Closed-form expressions for leave-one-out metrics

Leave-one-out cross-validation (often abbreviated LOO-CV) is a widespread tool in machine learning, which makes it possible to assess the performances of a model without resorting to an external test set. Its principle is straightforward: for each point of the n-sized training dataset, the model is trained and/or fitted on the $n-1$ other points, tested on the point left aside, and error metrics are computed on this prediction; then these metrics are aggregated over all n runs. We can give at least three interesting applications of LOO-CV in neutronics applications:

- **Model assessment**: any model used for interpolating HXS's in the core code should have been assessed after its fitting and/or training, in order to guarantee the quality of its predictions. In industry applications, it would be desirable that this validation doesn't rely on an external test set: otherwise, the computational effort dedicated to generate this set would be similar to this of generating the training data. LOO-CV makes this possible, by estimating the model's accuracy directly on the training outputs. In the same manner, several candidate models (having various modeling options, hyperparameters, etc.) can be compared this way without any external test set.
- **Outlier detection**: HXS data can be home to outliers, non-physical values which may greatly disturb machine learning models incorporating it. There is no unequivocal manner to categorize a datapoint as an outlier, as the variability of the data is rarely known in advance. LOO-CV offers a practical way to define outliers in an application-focused manner: if a few training points yield LOO errors above a predefined threshold, while all the other ones have much smaller errors, they are likely to be outliers and to harm the model if incorporated to it.
- **Training data pruning (downsampling)**: as explained in Section 4.1, for moderate-size HXS datasets, the number of training datapoints are a key for computational performance. Moreover, redundant training points can hinder the conditioning of kernel computations, for reasons exposed in Section 3.5. Therefore, it is often interesting to eliminate little-informative points from the training set. LOO-CV offers a convenient way to do so: the points with the smallest LOO errors are little useful to the model, as they can be very well predicted from the other ones. This way, one can incrementally *downsample* the training set, deleting at each iteration the best-approximated point. Preliminary experiments in this direction exhibited promising results: the size of the initial dataset could be divided by 2 with no significant loss of accuracy (see chapter 6.3 of [24]).

Naturally, the LOO-CV methodology can be applied to any machine learning model. But there is a specificity when using it with exact GPs: **leave-one-out errors can be computed with closed-form formulas, without the need of fitting the model at each run**. Such expressions can be found for instance in chapter 5.4.2 of [17] for SOGPs. We restate them here: if y_i is the i-th real data value, y_i^{LOO} its leave-one-out estimate – that is, the value obtained when fitting the model to $\mathbf{X}\backslash\mathbf{x_i}$ and $\mathbf{y}\backslash y_i$ and making a prediction on $\mathbf{x_i}$ – and v_i^{LOO} the predictive variance of this estimate, then y_i^{LOO} and v_i^{LOO} are given by :

$$y_i - y_i^{\mathrm{LOO}} = \left[(\mathbf{K}+\sigma^2\mathbf{I_n})^{-1}\mathbf{y}\right]_i / v_i^{\mathrm{LOO}} \tag{16}$$

$$v_i^{\mathrm{LOO}} = \left[(\mathbf{K}+\sigma^2\mathbf{I_n})^{-1}\right]_{ii} \tag{17}$$

where $[\mathbf{u}]_i$ denotes the i-th component of the vector $\mathbf{u}$, and $[\mathbf{M}]_{ii}$ the (i,i)-th coefficient of the matrix $\mathbf{M}$. Naturally, these expressions don't account for the (slight) modification of the optimal model parameters resulting from the removal of one data point from the training set.

Similar expressions are given in the PhD thesis from which this article is derived ([24]) for the PLMC and Lazy-LMC (which is essentially a subcase of the former). The latent processes of these models being conditionally independent, the LOO formulas of Equation (16) for SOGPs apply to them. It is then straightforward to recover the corresponding LOO estimates at the level of predicted tasks:

$$[\mathbf{U}]_{ij} - u_{ij}^{\mathrm{LOO}} = \left[(\mathbf{K_j}+\sigma_j^2\mathbf{I_n})^{-1}\mathbf{U_j^T}\right]_i / \vartheta_{ij}^{\mathrm{LOO}} \quad \forall j \in [1;q] \tag{18}$$

$$\vartheta_{ij}^{\mathrm{LOO}} = \left[(\mathbf{K_j}+\sigma_j^2\mathbf{I_n})^{-1}\right]_{ii} \quad \forall j \in [1;q] \tag{19}$$

$$\mathbf{Y} - \mathbf{Y}^{\mathrm{LOO}} = (\mathbf{Y}-\mathbf{HU}) + \mathbf{H}\,(\mathbf{U}-\mathbf{u}^{\mathrm{LOO}}) \tag{20}$$

$$\mathbf{v}^{\mathrm{LOO}} = \mathbf{H}\vartheta^{\mathrm{LOO}}\mathbf{H^T} \tag{21}$$

LOO-CV expressions are also given for the ICM in the chapter 5.4.2 of [24], but are omitted here because they would require to introduce the formalism of this model[33]. For the VLMC, no such expressions can be devised: indeed, the principle of this model is to replace some data-relative quantity of the GP posterior by a free-form distribution optimized in a black-box manner. The link between the training data and predictions, which underlies the above LOO expressions, is thus lost in the process; all information about the training data is stored implicitly in the model's parameters, as in a neural network.

We wished to assess whether these LOO estimates are good proxies for errors measured on an adequate external test set; this turned out to be the case for the PLMC and Lazy-LMC, but not for the ICM – see Appendix C for a quantitative exposition of this statement. In the latter case, the problem seems to arise from numerical issues, in a context of very ill-conditioned kernel matrices (see Sect. 3.5): the variance terms featured in the denominator of LOO error expressions (see Eqs. (16) and (18)) can be very small, and with the ICM they are strongly affected by conditioning issues, for being obtained by eigenvalue computations. A better

[33] Note that all discussed LOO-CV formulas were verified against LOO errors computed "by hand" – by explicit formation of the n data folds, and fitting (but not re-training) of the model to these folds – on several datasets, both real and synthetic. We found a perfect match between the values obtained by closed-form expressions and the empirical ones.

implementation of the ICM should thus be able to lift this issue quantifies how well the error metrics obtained with closed-form expressions match these computed on an external test set for our MOGP models.

5 Conclusions

In this article, we introduced the use of multi-output Gaussian processes (MOGP) to interpolate homogenized cross-sections (HXS) inside nuclear reactor neutronics codes. We compared several models of this family, discussed the main modeling choices which they offer and the best option is to retain for approximating HXS's with them. It was shown on a complete and realistic test-case that the desirable properties of these tools – tunable, long-ranged and smooth predictor, sparse interpolation support, *dimensionality reduction* leveraging the redundancy of the data – make it possible to improve on interpolation accuracy compared with the multilinear standard, with only a fraction of its training data (20 times fewer tabulated points required) and of its storage requirement (HXS library sizes reduced by two orders of magnitude). We also proved these models to be scalable to datasets of millions of HXS, with training durations of the order of the minute, and interpolation speeds shown analytically to match this of the multilinear method. Some of their convenient features – efficient and interpretable storage format, closed-form expressions of *leave-one-out errors* authorizing easy model assessment and training dataset pruning, etc. – were presented; their respective additional functionalities and limitations were compared in Table 1.

This collection of arguments seems to make MOGPs viable candidates for seamless integration into current neutronics codes, alleviating the burden associated with large HXS libraries in today's best-estimate calculations. The next step for assessing this methodology will be to implement it in an actual core code, and validate on a variety of benchmarks that it doesn't hinder the accuracy of simulations. The models should also be tested on more diverse tests-cases: other reactor technologies, more HXS variables (moderator density, xenon concentration, etc.), accidental conditions... Incidentally, one issue not addressed in this work is this of *discrete variables*, such as the type of control rod inserted in the assembly: while kernels able to handle such inputs exist [47], modeling HXS data with them has not been tested yet[34].

Apart from validation, the main developments to bring to this line of work are *uncertainty quantification* and *active learning*. Uncertainty quantification is a growing subject in neutronics, and Gaussian processes would be valuable additions to UQ pipelines: they make it possible to propagate uncertainties trivially[35], non-linearly and explicitly, contrary to other machine learning models like neural networks for which uncertainty propagation is an ongoing field of research[36] [48]. We showed in the present article that although sensitive to several factors, good-quality variance estimates could be obtained with MOGPs on HXS data; much more thorough investigation of their statistical properties should now be undertaken. *Active learning*, for its part, refers to the data-informed construction of an optimal training dataset; such methodologies could further reduce the computational burden of generating HXS libraries, and make it possible to parametrize HXS's with many more variables for instance. First results were obtained in this direction in the PhD thesis associated with this work [24], but much more ambitious developments could be envisioned thanks to the large amount of literature dedicated to active learning with Gaussian processes [49].

Glossary

ANN: Artificial Neural Network

BPR: Bayesian Polynomial Regression

EVR: Expected Variance Ratio

GP: Gaussian Process

HXS: Homogenized Cross-Section

ICM: Intrinsic Coregionalization Model

LMC: Linear Model of Coregionalization

LOO-CV: Leave-One-Out Cross-Validation

MLI: Multi-Linear Interpolation

MLL: Marginal Log-Likelihood

MOGP: Multi-Output Gaussian Process

PCA: Principal Components Analysis

PLMC: Projected Linear Model of Coregionalization

PVA: Predictive Variance Adequacy

R2: coefficient of determination

RMSE: Root Mean Squared Error

SOGP: Single-Output Gaussian Process

[34] In the worst-case scenario, if these discrete kernels proved unsatisfactory, one could still model discrete variables as *output tags* rather than *input variables*: one HXS with three possible values of control rod insertion would be seen instead as three correlated outputs of the same MOGP.

[35] One simply has to replace the noise matrix of the MOGP estimator by the covariance matrix of the propagated uncertainties.

[36] The reason for this is that with such models, there is no explicit link between the training data and predictions: all information about the former is contained in the model's weights. Therefore, one cannot apply a perturbation to the training data and observe its effect on the model's outputs.

SVD: Singular Values Decomposition

UQ: Uncertainty Quantification

VLMC: Variational Linear Model of Coregionalization

Math Symbols:

- n : number of training datapoints
- p : number of regression tasks (modeled HXS's)
- q : number or *latent processes* (see Sect. 2.2)
- d : dimension of the inputs, i.e number of variables along which HXS's are parametrized
- $\mathbf{H}$: *mixing matrix* of the LMC model (see Sect. 2.2), of size $p \times q$
- $\mathbf{U}$: low-dimensional summary of the data seen by the latent processes (see paragraph 4.1), of size $q \times n$
- $\mathbf{X}$: matrix containing all training data inputs, of size $d \times n$
- $\mathbf{Y}$: matrix containing all training data outputs, of size $p \times n$
- $\mathbf{y}$: size-n vector containing all training data outputs in the case of a single-output model
- σ^2 : noise term of a GP, added to the diagonal of the kernel matrix
- σ_B^2 : lower bound enforced by σ^2 in some implementations
- x_* : input point at which a prediction is made
- k : kernel function of the GP
- $\mathbf{k_{X*}}$: vector $k(\mathbf{X}, \mathbf{x}_*)$ $(= [k(\mathbf{x_i}, \mathbf{x}_*)]_{1 \leq i \leq n})$
- $\mathbf{K}$: matrix $k(\mathbf{X}, \mathbf{X})$ $(= [k(\mathbf{x_i}, \mathbf{x_k})]_{1 \leq i \leq n}^{1 \leq j \leq n})$
- k_{**} : scalar $k(\mathbf{x}_*, \mathbf{x}_*)$

Acknowledgments

APOLLO3® is registered trademarks of CEA.

Funding

O. Truffinet, K. Ammar and N. Gérard-Castaing thank EDF and Framatome for partial financial support.

Conflicts of interest

The authors have nothing to disclose.

Data availability statement

Data associated with this article cannot be disclosed for reasons of confidentiality.

Author contribution statement

Conceptualization and methodology, O. Truffinet, K. Ammar, B. Bouriquet and J.-P. Argaud; Software, O. Truffinet and N. Gérard Castaing.; Resources, O. Truffinet and K. Ammar; Investigation, Data Curation, Validation, Visualization and Writing – Original Draft Preparation, O. Truffinet; Supervision and Writing – Review & Editing, K. Ammar, B. Bouriquet and J.-P. Argaud.

References

1. A. Weiser, E. Zarantonello, A note on piecewise linear and multilinear table interpolation in many dimensions, Math. Comput. **50**, 189 (1988)
2. S. Sánchez-Cervera et al., Optimization of multidimensional cross-section tables for few-group core calculations, Ann. Nucl. Energy **69**, 226 (2014)
3. V. Zimin, A. Semenov, Building neutron cross-section dependencies for few-group reactor calculations using stepwise regression, Ann. Nucl. Energy **32**, 119 (2005)
4. P. Bokov, Automated few-group cross-section parameterization based on quasi-regression, Ann. Nucl. Energy (2009)
5. D. Botes, P. Bokov, Polynomial interpolation of few-group neutron cross sections on sparse grids, Ann. Nucl. Energy **64**, 156 (2014)
6. E.A. Szames, Amélioration du modèle reconstruction des sections efficaces dans un code de calcul de neutronique à l'échelle cœur, Ph.D. thesis, Université Paris-Saclay 2020
7. J.P. Berrut, L.N. Trefethen, Barycentric lagrange interpolation, SIAM Rev. **46**, 501 (2004)
8. K. Tan et al., The assembly homogeneous fewgroup constants generation method for PWR based on machine learning, Ann. Nucl. Energy **165**, 108772 (2022)
9. V.L. Nicolas Martin Zachary Prince, M. Tano-Retamales, Deep learning for multigroup cross-section representation in two-step core calculations, Nucl. Sci. Eng. **197**, 1406 (2023)
10. Y.M. Chan, J. Dufek, A deep-learning representation of multi-group cross sections in lattice calculations, Ann. Nucl. Energy **195**, 110123 (2024)
11. S. Dzianisau, D. Lee, Application of artificial neural network for assembly homogenized equivalence parameter generation, Prog. Nucl. Energy **173**, 105285 (2024)
12. D. Tomatis, A multivariate representation of compressed pin-by-pin cross sections, EPJ Nucl. Sci. Technol. **7**, 8 (2021)
13. M. Yamamoto, T. Endo, A. Yamamoto, Compression of cross-section data size for highresolution core analysis using dimensionality reduction technique, Nucl. Sci. Eng. **195**, 33 (2021)
14. B. Whewell, R.G. McClarren, Data reduction in deterministic neutron transport calculations using machine learning, Ann. Nucl. Energy **176**, 109276 (2022)
15. G. Hua, Y. Li, S. Wang, PWR pinhomogenized cross-sections analysis using big-data technology, Prog. Nucl. Energy **121**, 103228 (2020)
16. D. Tomatis, A. Dall'Osso, Compression of 3D pin-by-pin burnup data, Ann. Nucl. Energy **136**, 107058 (2020)
17. C.E. Rasmussen, C.K.I. Williams, *Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning* (MIT Press, Cambridge, Mass., 2006), p. 248
18. M.A. Alvarez, L. Rosasco, N.D. Lawrence, Kernels for vector-valued functions: A review, Found. Trends Mach. Learn. **4**, 195 (2012)
19. E.V. Bonilla, K. Chai, C. Williams, Multitask Gaussian process prediction, in *Advances in Neural Information Processing Systems*, edited by J. Platt et al. (Curran Associates, Inc., 2007), Vol. 20
20. M. van der Wilk et al., A framework for interdomain and multioutput Gaussian processes (2020). arXiv:2003.01115
21. O. Truffinet et al., Exact and general decoupled solutions of the LMC Multitask Gaussian Process model (2024). arXiv:2310.12032
22. T.W. Evans, P.B. Nair, Exploiting structure for fast kernel learning, in *Proceedings of the 2018 SIAM International Conference on Data Mining (SDM)* (2018), pp. 414–422

23. S. Van Vaerenbergh et al., Fixed-budget kernel recursive least-squares, in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing* (2010), pp. 1882–1885
24. O. Truffinet, Machine learning methods for cross-section reconstruction in full-core deterministic neutronics codes, Ph.D. thesis, Université Paris-Saclay, 2024
25. B. Rakitsch et al., It is all in the noise: Efficient multi-task Gaussian process inference with structured residuals, in *Advances in Neural Information Processing Systems*, edited by C. Burges et al. (Curran Associates, Inc., 2013), Vol. 26
26. J. Hensman, A. Matthews, Z. Ghahramani, Scalable variational Gaussian process classification, in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, edited by G. Lebanon, S.V.N. Vishwanathan (PMLR, San Diego, California, USA, 2015), Vol. 38, pp. 351–360.
27. H. Liu et al., Learning multitask Gaussian process over heterogeneous input domains, IEEE Trans. Syst. Man Cybern. Syst. **53**, 6232 (2023)
28. M. Jankowiak, G. Pleiss, J. Gardner, Parametric Gaussian process regressors, in *International Conference on Machine Learning* (PMLR, 2020), pp. 4702–4712
29. W. Bruinsma et al., Scalable exact inference in multi-output Gaussian processes, in *ICML'20: Proceedings of the 37th International Conference on Machine Learning*, edited by H. Daume, A. Singh (2020)
30. M. Gu, W. Shen, Generalized probabilistic principal component analysis of correlated data, J. Mach. Learn. Res. **21**, 428 (2020)
31. G. Wahba, *Spline models for observational data* (1990). https://doi.org/10.1137/1.9781611970128
32. O. Truffinet et al., An EIM-based compressionextrapolation tool for efficient treatment of homogenized cross-section data, Ann. Nucl. Energy **185**, 109705 (2023)
33. N. Dige, U. Diwekar, Efficient sampling algorithm for large-scale optimization under uncertainty problems, Comput. Chem. Eng. **115**, 431 (2018)
34. Y. Li et al., PWR few-group constants parameterization analysis, Prog. Nucl. Energy **88**, 104 (2016)
35. H. Wendland, Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree, Adv. Comput. Math. **4**, 389 (1995)
36. A. Godfrey, Vera Core Physics – Benchmark Progression – Problem Specifications, Tech. Rep., Consortium for Advanced Simulation of LWRs (Oak Ridge National Laboratory, 2014)
37. P. Mosca et al., APOLLO3®: Overview of the new code capabilities for reactor physics analysis, in *M&C 2023* (2023)
38. E. Szames et al., Few-group cross sections modeling by artificial neural networks, EPJ Web Conf. **247**, 06029 (2021)
39. D.P. Kingma, J. Ba, Adam: A method for stochastic optimization (2017). arXiv:1412.6980
40. A. Paszke et al., PyTorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2019), Vol. 32
41. J.R. Gardner et al., GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration, in *Advances in Neural Information Processing Systems* (2018)
42. N. Martin, M. Riedmann, J. Bigot, Latest developments in the ARTEMIS TM core simulator for BWR steady-state and transient methodologies, in *MC2017* (2017)
43. M.E. Tipping, Sparse bayesian learning and the relevance vector machine, J. Mach. Learn. Res. **1**, 211 (2001)
44. J.P. Boyd, The uselessness of the Fast Gauss Transform for summing Gaussian radial basis function series, J. Comput. Phys. **229**, 1311 (2010)
45. K. Cutajar et al., Preconditioning kernel matrices, in *Proceedings of the 33rd International Conference on Machine Learning*, edited by M.F. Balcan, K.Q. Weinberger (PMLR, New York, New York, USA, 2016), Vol. 48, pp. 2529–2538
46. P. Monasse, Extraction of the level lines of a bilinear image, Image Process. Line **9**, 205 (2019)
47. P. Saves et al., A mixed-categorical correlation kernel for Gaussian process, Neurocomputing **550**, 126472 (2023)
48. J. Gawlikowski et al., A survey of uncertainty in deep neural networks, Artif. Intell. Rev. **56**, 1513 (2023)
49. H. Liu, Y.-S. Ong, J. Cai, A survey of adaptive sampling for global metamodeling in support of simulation-based complex engineering design, Struct. Multidiscip. Optim. **57**, 393 (2018)

Cite this article as: Olivier Truffinet, Karim Ammar, Jean-Philippe Argaud, Nicolas Gérard Castaing, Bertrand Bouriquet. Multi-output Gaussian processes for the reconstruction of homogenized cross-sections, EPJ Nuclear Sci. Technol. **11**, 61 (2025). https://doi.org/10.1051/epjn/2025055

Appendix A: Experimental results on the finely-discretized dataset

Here we present our experimental results on our large dataset (12 assemblies containing $3.5 \cdot 10^5$ HXS's each, see Sect. 3.1); we left them aside in Section 3.4, as they are very similar to these of Table 5 but require a few additional explanations. These results are summarized in Table A.1. Here are some points to specify:

- The ICM is absent from this table, because as mentioned in Section 3.3, current implementations use a full-rank correlation matrix between all p tasks [25], which incurs computational costs in $O(p^3)$, intractable in this case. This could be remedied in the future by using a rank-q correlation matrix instead, $\mathbf{K_f} = \mathbf{H}\mathbf{H}^\mathbf{T}$ which would yield the same correlation structure as with our other LMC models.
- The MLI baseline is also absent, as we wanted to avoid the generation, storage and manipulation of such a dataset on full Cartesian Grids: it would have weighted about 200 gigabytes.
- The "best SOGP" baseline is replaced by "Training-less SOGPs": this is because training millions of SOGP models on a standard machine, at the rythm of a few seconds per model, would be infeasible. These training-less SOGPs are the same model that wraps the latent processes of our Lazy-LMC: their kernel is the already-introduced, parameter-less spline kernel, and their few additional parameters have been neutralized (see Tab. 2).
- The metrics assessing the calibration of predictive variance are missing for the PLMC. This is because predictive variance could not be computed in this setting with our current implementation of this model: it requires the formation of a full $p \times p$ noise matrix, which causes a memory error for large p. This is only a matter of implementation: only the diagonal of this matrix is required for variance computation, so these coefficients could very well be computed alone without overloading memory.
- It may be surprising that the $R2$ coefficients of the PLMC and VLMC are respectively subpar and catastrophic, while their other error metrics are excellent. This relates to an observation already made in Section 3.5: **if the training data of each regression task is not normalized to have variance $\simeq$1, our trainable MOGP models tend to neglect the low-variance tasks**. In other terms, they are "lazy" with these tasks: they achieve small errors on them, which is why all other metrics than the $R2$ remain flawless, but these errors are in fact significant compared to the small variability of the modeled function. Once again, this advocates for the use of standard deviation-based normalization of each HXS before model training. More recent experiments confirmed that this choice indeed restores all $R2$ coefficients to excellent values.

These remarks being made, it can be seen that **all numerical values are very close to these of Table 5: our models perform just as well in the many-HXS setting as in the standard one**. Apart from this, the main observation is that **the GPU training of our models – in particular the PLMC – is very fast: two minutes are enough to fit a PLMC to hundreds of thousands of HXS's**, making it fully ready for the big-data setting. Other minor remarks can be made: BPR performs less well on this dataset, which strengthens our intuition that GPs are more reliable than polynomial methods for modeling diverse HXS datasets; and both training-less models (Lazy-LMC and training-less SOGPs) still perform remarkably well on this test-case, although the gap between the Lazy-LMC and VLMC is larger than on the smaller dataset, as it was already observed in Figure 3b.

Appendix B: Optimal MOGP model size for neutronics calculations

The experiments of Appendix A showed that training a single model for very many HXS's was feasible. However, it is also somewhat troublesome: it can be slow, even with GPU mini-batch training (more than 2 hours per assembly with the VLMC); the memory footprint can exceed the capacity of a single GPU; training is less stable, as illustrated in Figure 3b, etc. **One should therefore wonder whether this methodology is really appropriate, or whether fitting smaller models to subsets of the data should be preferred**. Eventual gains of the full-model option would be storage savings – by mutualizing latent processes across more outputs, the storage of the data defining them can be spared – and computational savings (likewise, the interpolation of latent processes would be mutualized). But in large-data and low-q settings, these gains can be seen to be negligible.

Consider for instance our finely-discretized dataset, which neutronics data is defined at the level of the fuel pin; **assume that we split each assembly-wise dataset between its n_c fuel cells, and define one PLMC or Lazy-LMC model per cell, keeping the same values of q and n for each**. Each of these models now has $p' = p/n_c$ tasks. According to Section 4.1, the cost of making predictions with each one is $2p'q + n(c_k + 2q)$, so the total cost for *all* of them is $n_c \times (2p'q + n(c_k + 2q)) = 2pq + n_c\, n(c_k + 2q)$. By contrast, the cost of predictions with a single assembly-wise model is $2pq + n(c_k + 2q)$. The question is thus the relative size of $2pq$ and $n_c\, n(c_k + 2q)$.

In the setting of our previous experiments, we have $p = 3.5 \cdot 10^5$, $q = 16$, $n_c = 45$ (one assembly with 17×17 cells and 1/8 symmetry), $n = 201$ and $c_k = 30$. Therefore, $2pq \simeq 1.12 \cdot 10^7$, while $n_c\, n(c_k + 2q) \simeq 5.6 \cdot 10^5$: **the lifting of the low-dimensional interpolated values to the "real" HXS space remains 20 times more costly than the interpolation of the latent processes**, even if there are now 45 times more latent processes to interpolate (because the model was split into 45 smaller ones). This lifting operation doesn't depend on the allocation of HXS's into one or several models, only on the retained value of q. The exact same argument can be given about storage requirements: they remain dominated by the

Table A.1. Performance metrics on all assemblies of the finely-discretized dataset, for three of the studied MOGP models (all with $q = 16$ latent processes) and two baselines. The best and worst value for each metric are highlighted in green and red respectively. Definitions of the metrics are given in Section 3.2. The training time T_{train} is defined at the level of one assembly. PLMC and VLMC were trained on GPU, with minibatch training in the latter case. The α-CI and PVA metrics are missing for the PLMC are missing because predicted variance was not computed with this model: our current implementation requires the formation of a full $p \times p$ noise matrix for this, which causes a memory error in this setting.

Model	$R2$	RMSE	Err_{max}	$\text{Err}^{\Sigma}_{\text{mean}}$ (cm^{-1})	$\text{Err}^{\Sigma}_{\text{max}}$ (cm^{-1})	α-CI	PVA	T_{train} (s)	$\mathcal{R}$
VLMC	0.185	$1.86 \cdot 10^{-3}$	0.189	$2.04 \cdot 10^{-4}$	0.543	1.00	−5.90	8,010	266
PLMC	0.984	$2.43 \cdot 10^{-3}$	0.169	$1.69 \cdot 10^{-4}$	0.423	–	–	122	266
Lazy-LMC	0.999	$2.97 \cdot 10^{-3}$	0.145	$1.75 \cdot 10^{-4}$	0.469	1.00	−11.5	$5 \cdot 10^{-4}$	266
BPR	0.997	$1.19 \cdot 10^{-1}$	0.758	$1.78 \cdot 10^{-4}$	0.348	0.987	−1.48	6310	13
Training-less SOGP	0.999	$2.86 \cdot 10^{-3}$	0.111	$1.24 \cdot 10^{-4}$	0.434	0.999	−6.03	12 200	21

storage of the $p \times q$ matrix $\mathbf{H}$, next to which the few parameters of each latent process and the training caches $\mathbf{C}$ of Section 4.1 are negligible.

Therefore, **in the many-HXS setting, typically if pin-by-pin homogenization is performed, there is no clear benefit to gathering all HXS's from a dataset into a single model: pin-wise models can be used instead without noticeable loss in computation speed or storage space**. The question to be asked before deciding on a grouping of HXS's is the relative size of the mixing matrix $\mathbf{H}$ and of the quantities defining the latent processes: as long as the former dominates, there is no harm in splitting a large model into smaller one[37]. Incidentally, there can be practical issues when gathering some categories of HXS's into a single model. For instance, **if pin-by-pin burnup values are used and HXS from all pins are modeled by the same MOGP, the tasks of this model will be trained with differing training inputs $\mathbf{X}$**, as HXS's from different pins will be exposed to different burnups. This at least requires dedicated implementations, and may not be readily available (or available at all) for all models, as stated in Table 1.

Appendix C: Adequacy of leave-one-out cross-validation for model assessment

In Section 4.3, we gave closed-form expressions for the LOO-CV errors of our models, and indicated their possible uses in neutronics calculation chains. One of them is to assess the accuracy of trained models without resorting to an external test set. However, we would like to ensure that this methodology is trustworthy, that is, that the error metrics we obtain with LOO-CV are consistent with these obtained on test sets. This is the case if two conditions are met:

- **LOO-based error metrics are well-correlated with these measured on the test set, across various model instances**. Indeed, it would be problematic if they designated a specific model declination (some combination of modeling choices) as "good", but the predictions of this model turned out to be unsatisfactory.
- **LOO-based error metrics are well-calibrated with the ones measured on the test set**. By this, we mean that the values of LOO-based metrics should always be close to these of their test-based counterparts: one must be able to expect some level of accuracy when applying the model to unseen data.

The only manner in which we can assess these criteria experimentally is by computing LOO estimates to test-set errors for various model declinations (models spanning the full range of the modeling choices discussed in this article: mean function, kernel function, number of latent processes, etc.). This was done in all grid searches of Section 2.5, and in the experiments of Section 3.3 on the number of latent processes. We thus dispose of a set of experiments with various model declinations, each yielding a series of error metrics $\mathcal{M}_{ij}$ computed on the test dataset – where i denotes the metric (RMSE, Err_{mean}, PVA, etc.), and j the tested model – and their leave-one-out counterparts $\mathcal{M}^{\text{LOO}}_{ij}$ computed on the training dataset. These metrics are RMSE, Err_{mean}, Err_{max} and $q99$; $q99$ is the quantile of L1 errors at level 99%, added here for additional information about large errors. From this we compute the ratios $r_{ij} = \mathcal{M}^{\text{LOO}}_{ij}/\mathcal{M}_{ij}$, and the correlations $c_i = \text{Corr}_j(\mathcal{M}^{\text{LOO}}_{ij}, \mathcal{M}_{ij})$. For each metric $\mathcal{M}_i$, we summarize its computed ratios r_{ij} (of which there are as many as models tested) by a few of their statistics: minimum, maximum, mean and standard deviation. Results of this analysis are displayed below in Tables C.1 for the PLMC and C.2 for the ICM; each one corresponds to the grid search previously performed on these models, which feature respectively 561 and 300 tested model declinations, ensuring the meaningfulness of the computed statistics. As stated above, these statistics were also computed on the experiments of Section 3.3 studying the variations of

[37] On the contrary, gathering many disparate HXS's into a single model may increase the value of q yielding the desired accuracy, which computation times and memory requirements scale linearly with.

Table C.1. Analysis of the agreement between test-based and LOO-based accuracy metrics for the PLMC. For any metric $\mathcal{M}_i$, $\mathcal{M}_{ij}$ denotes its value for the j-th model declination, $\mathcal{M}_{ij}^{\text{LOO}}$ its LOO counterpart, and $r_{ij} = \mathcal{M}_{ij}^{\text{LOO}}/\mathcal{M}_{ij}$.

Metric $\mathcal{M}_i$	$\text{Corr}_j(\mathcal{M}_{ij}^{\text{LOO}}, \mathcal{M}_{ij})$	$\max(r_{ij})$	$\min(r_{ij})$	$\text{mean}(r_{ij})$	$\text{std}(r_{ij})$
RMSE	0.9560	1.20	0.394	0.903	0.1430
Err_{mean}	0.9360	1.23	0.384	0.886	0.1470
Err_{max}	0.8870	2.48	0.367	0.842	0.2040
$q99$	0.9660	1.40	0.383	0.943	0.1530

Table C.2. Analysis of the agreement between test-based and LOO-based accuracy metrics for the ICM. For any metric $\mathcal{M}_i$, $\mathcal{M}_{ij}$ denotes its value for the j-th model declination, $\mathcal{M}_{ij}^{\text{LOO}}$ its LOO counterpart, and $r_{ij} = \mathcal{M}_{ij}^{\text{LOO}}/\mathcal{M}_{ij}$. A few models have undefined PVA values because of some near-zero variance estimate, which are not taken into account in the statistics.

Metric $\mathcal{M}_i$	$\text{Corr}_j(\mathcal{M}_{ij}^{\text{LOO}}, \mathcal{M}_{ij})$	$\max(r_{ij})$	$\min(r_{ij})$	$\text{mean}(r_{ij})$	$\text{std}(r_{ij})$
RMSE	0.1680	3.92	9.22e–03	0.849	0.544
Err_{mean}	0.1740	3.88	8.19e–03	0.864	0.516
Err_{max}	0.2060	2.57	1.08e–02	0.769	0.432
$q99$	0.1520	4.96	8.32e–03	0.874	0.606

Table C.3. Analysis of the agreement between test-based and LOO-based accuracy metrics for the Lazy-LMC (not measured on a grid search, but on the experiment of Fig. 3a). For any metric $\mathcal{M}_i$, $\mathcal{M}_{ij}$ denotes its value for the j-th model declination, $\mathcal{M}_{ij}^{\text{LOO}}$ its LOO counterpart, and $r_{ij} = \mathcal{M}_{ij}^{\text{LOO}}/\mathcal{M}_{ij}$.

Metric $\mathcal{M}_i$	$\text{Corr}_j(\mathcal{M}_{ij}^{\text{LOO}}, \mathcal{M}_{ij})$	$\max(r_{ij})$	$\min(r_{ij})$	$\text{mean}(r_{ij})$	$\text{std}(r_{ij})$
RMSE	0.989	1.04	0.671	0.819	0.102
Err_{mean}	0.985	1.02	0.610	0.768	0.124
Err_{max}	0.963	1.13	0.705	0.998	0.0947
$q99$	0.991	1.02	0.646	0.807	0.109

MOGP performance with q: results are fully consistent with these obtained on grid searches, but less relevant as the number of items is small (for each model, 18 declinations differing only by their value of q). We still display them for the Lazy-LMC in Table C.3, by lack of an alternative: the only modeling choice left in this model is indeed the number of latent processes.

Results are clear: **LOO estimates are very good proxies of test set errors for PLMC and Lazy-LMC, but much less for ICM**. For the first two, the ratios r_{ij} are close to 1 on average, and contained in the range $[0.36; 2.48]$ meaning that any LOO metric is at worse 3 times smaller or larger than its test-data counterpart; moreover, the correlations between LOO and non-LOO metrics are always very close to 1. The situation of the ICM is much more troublesome: correlations between metrics and their LOO counterparts are small ($\simeq$20%), and ratios between these quantities can take values smaller than 1% – although their means are still rightfully close to 1. Taking a closer look at the experimental values, it appears that this discrepancy is due to a few very ill-performing model instances, which LOO estimates are nonetheless satisfactory. As in Section 3.5, this suggests some subpar behavior of linear algebra operations in our implementation of this model: while it usually yields consistent and satisfying results[38], it sometimes breaks down on edge cases – computation of small variance terms in contexts of ill-conditioned, poorly-performing model instances. Recall indeed that our LOO expressions (Eqs. (16) and (18)) involve variance terms, which can be very small for little-noisy HXS's and particularly sensitive to numerical issues (see Sect. 3.5).

Lastly, let us mention that the same tests as in Tables C.1–C.3 were performed on the metrics assessing the quality of variance assessment, α-CI and PVA. The results, given in the Section 5.5 of [24], are catastrophic: correlations are tiny, and ratios are erratic. This is another indication of the malfunction of MOGP predictive variance in our application when no specific attention is paid to it. The same experiments should now be undertaken after fixing predictive variances with the solutions suggested in Section 3.5, in order to definitely validate the reliability of these LOO-CV estimates.

[38] In particular, our LOO-CV expressions for this model, given in Section 5.5 of [24], were validated by comparison with empirical LOO values on several datasets; perfect matches were observed in each case.