Supplementary Information of "Global Ranking of the Sensitivity of Interaction Potential Contributions within Classical Molecular Dynamics Force Fields"

Wouter Edeling[1], Maxime Vassaux[2], Yiming Yang[3], Shunzhou Wan[4], Serge Guillas[3], and Peter V. Coveney*[4,5,6]

[1]Scientific Computing Group, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands.
[2]Université de Rennes, CNRS, Institut de Physique de Rennes, UMR 6251, Rennes, 35000, France
[3]Department of Statistical Science, University College London, London, United Kingdom.
[4]Centre for Computational Science, University College London, London, United Kingdom.
[5]Advanced Research Computing Centre, University College London, London, United Kingdom.
[6]Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands.

September 22, 2023

S1 Computational setup

To generate the training data for the neural-network / GP-based active subspace methods, we conducted three separate UQ campaigns to sample the molecular dynamics systems described in the introduction: one for materials property prediction and two concerned with absolute and relative binding free energy calculations. The sampling plans for the three UQ campaigns were handled by the open-source Python UQ library EasyVVUQ [1]. The library facilitates the sampling of the parametric space, the generation of the samples and the aggregation of the simulation data for analysis. For each application, we generated and simulated hundreds of samples using Monte Carlo sampling and ensemble averaged over the replicas which were generated by different random seeds. A sample here involved the execution of the protocol relying on ensemble MD simulation to obtain the prediction of a given QoI (elastic moduli and free energies). The number of replicas per ensemble is determined independently for each application based on the convergence of the confidence interval of the predicted value of the QoI for a fixed parametric configuration. This resulted, as shown in Table 1 of the main article, in 20 replicas for the epoxy materials simulations, 25 for the ESMACS calculations, and 5 for the TIES calculations. The computational requirements associated with the execution of one sample varied from one application and/or protocol to the other as follows:

- The materials properties calculations were performed on the SuperMUC-NG supercomputer at Leibniz Rechenzentrum in Garching, Germany using 288 cores per replica for 2 hours. The production of one ensemble MD (comprising 20 replicas) calculation required 11,520 core-hrs and a total of 5.7M core-hrs for the complete UQ campaign (comprising 500 samples).
- The ESMACS binding free energy calculations were performed on SuperMUC-NG as well on 48 cores per replica for 13 hours. The production of one ensemble-MD (comprising 25 replicas) calculation resulted in 14,400 core-hrs and a total of more than 10M core-hrs for the complete UQ campaign (comprising 736 samples).

- The TIES binding free energy calculations were performed on the UK national supercomputer ARCHER2 on 1,664 cores per replica (128 cores for each of the 13 states in one alchemical transition) for 4 hours for the ligand-protein complex and on 156 cores per replica for 3 hours for the ligand part of the simulation. The production of one ensemble-MD (comprising 5 replicas) calculation resulted in 35,620 core-hrs and a total of 18M core-hrs for the complete UQ campaign (comprising 488 samples).

In both the UQ campaigns targeting binding free energy calculations, the computational workflows were subject to a occurrence of sample simulation failures, namely a failure rate of 26.4% for ESMACS and 51.2% for the TIES application. The epoxy application did not experience any failure to converge. The TIES protocol requires not only the independent simulation of ensembles of replicas (as for the ESMACS protocol), but also independent simulations at different stages of the integration path for both the ligand-only and ligand-protein complex. The failure of a single independent simulation causes the failure of the binding free energy calculation. We used 1 fs and 2 fs timesteps for LAMMPS and NAMD studies, respectively (Tables S2, S3 and S5). Different time steps were applied because these are the standard ones used by us in our extensive scientific work in these two domains, and are based on our own observations of stability of the two integrators in such contexts. The time step selected is no doubt very important as its magnitude controls the stability of the integrators used, and is likely to have an impact on the stability of these codes under such stochastic variations in parameters. In the TIES study, atoms in the alchemical region can get closer than that in normal simulations, making it even more sensitive to the changes of parameters in the non-bonded interactions. This explains why a higher failure rate is observed in TIES than that in ESMACS.

The materials properties calculations for bulk epoxy resins are much less sensitive to such failures, as in fact all 500 samples converged. The density of the materials simulations in comparison to protein-ligand simulation is higher. The higher viscosity of the system may in turn stabilise the molecular system when it comes to potentially unstable dynamics.

S2 Deep active subspaces

Here we will briefly outline deep active subspaces, developed by [2], and further studied by us [3]. In Section S3 our Gaussian-process based approach is outlined.

The Deep-Active Subspace (DAS) method makes use of the fact that the active subspace has the form of the activation of a hidden layer in a neural network, if the activation function Φ were linear: $\mathbf{y} = \Phi\left(W_1^T\mathbf{x}\right) = W_1^T\mathbf{x}$. Hence the overall idea is to mimic the active subspace method via a linear encoder in the first hidden layer of a feed-forward neural network. This so-called DAS layer is connected to the (high-dimensional) input layer $\mathbf{x}$, has weight matrix $W_1 \in \mathbb{R}^{D\times d}$, and has $d \ll D$ neurons. As the eigenvectors U_1 form an orthonormal basis for $\mathbf{y}$, W_1 should be orthonormal as well. Orthonormality can be built into the architecture of the network, by first defining a non-orthonormal basis $Q \in \mathbb{R}^{D\times d}$. These are the independent variables over which the DAS layer is optimized via the customary combination of stochastic gradient descent and back propagation [4]. Each orthonormal vector $\mathbf{w}_i \in \mathbb{R}^D$ is created from the column vectors $\mathbf{q}_i \in \mathbb{R}^D$ of Q via Gram-Schmidt orthogonalization, i.e.;

$$W_1(Q) = \left[\frac{\mathbf{w}_1(\mathbf{q}_1)}{\|\mathbf{w}_1(\mathbf{q}_1)\|_2} \quad \frac{\mathbf{w}_2(\mathbf{q}_1,\mathbf{q}_2)}{\|\mathbf{w}_2(\mathbf{q}_1,\mathbf{q}_2)\|_2} \quad \cdots \quad \frac{\mathbf{w}_d(\mathbf{q}_1,\mathbf{q}_2\cdots,\mathbf{q}_d)}{\|\mathbf{w}_d(\mathbf{q}_1,\mathbf{q}_2\cdots,\mathbf{q}_d)\|_2}\right]. \tag{1}$$

The well-known Gram-Schmidt vectors are given by;

$$\mathbf{w}_i = \mathbf{q}_i - \sum_{j=1}^{i-1}\left(\frac{\mathbf{w}_j^T\mathbf{q}_i}{\mathbf{w}_j^T\mathbf{w}_j}\right)\mathbf{w}_j, \quad i = 1,\cdots,d. \tag{2}$$

Stochastic gradient descent requires the gradient of the (squared) loss function L with respect to Q,

which can be expressed as;

$$\frac{\partial L}{\partial Q} = \begin{bmatrix} \left\langle \frac{\partial L}{\partial W_1}, \frac{\partial W_1}{\partial q_{11}} \right\rangle_F & \cdots & \left\langle \frac{\partial L}{\partial W_1}, \frac{\partial W_1}{\partial q_{1d}} \right\rangle_F \\ \vdots & \ddots & \vdots \\ \left\langle \frac{\partial L}{\partial W_1}, \frac{\partial W_1}{\partial q_{D1}} \right\rangle_F & \cdots & \left\langle \frac{\partial L}{\partial W_1}, \frac{\partial W_1}{\partial q_{Dd}} \right\rangle_F \end{bmatrix}. \tag{3}$$

see [3]. Each Frobenius inner product $\langle \cdot, \cdot \rangle_F$ contains $\partial L / \partial W_1$ which is available through standard back propagation. The $\partial W_1 / \partial q_{ij}$ matrices ($i = 1, \cdots, d,\ j = 1, \cdots, D$) can also be computed as Gram-Schmidt vectors are differentiable. One could rely on automatic differentiation to compute $\partial L / \partial Q$ [2], or as we do here, use (3) with an analytical expression for the $\partial W_1 / \partial q_{ij}$ terms, derived in [3], and displayed here for convenience:

$$\frac{\partial \mathbf{w}_1}{\partial \mathbf{q}_1} =: D_{11} = I_D \tag{4}$$

$$\frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_i} =: D_{ii} = D_{i-1,\,i-1} - \frac{\mathbf{w}_{i-1}\mathbf{w}_{i-1}^T}{\mathbf{w}_{i-1}^T\mathbf{w}_{i-1}}, \quad i > 1, \tag{5}$$

$$\frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_k} = \sum_{j=1}^{i-1} D_{ij} \frac{\partial \mathbf{w}_j}{\partial \mathbf{q}_k}, \quad i \neq k, \quad i > k, \tag{6}$$

$$D_{ij} := -\frac{\partial}{\partial \mathbf{w}_j} \left[\left(\frac{\mathbf{w}_j^T \mathbf{q}_i}{\mathbf{w}_j^T \mathbf{w}_j} \right) \mathbf{w}_j \right] =$$

$$- \left[\frac{1}{\mathbf{w}_j^T \mathbf{w}_j}\ \mathbf{w}_j \mathbf{q}_i^T - \frac{2\mathbf{w}_j^T \mathbf{q}_i}{(\mathbf{w}_j^T \mathbf{w}_j)^2}\ \mathbf{w}_j \mathbf{w}_j^T + \frac{\mathbf{w}_j^T \mathbf{q}_i}{\mathbf{w}_j^T \mathbf{w}_j}\ I_D \right], \quad i \neq j, \quad i > j, \tag{7}$$

$$\frac{\partial}{\partial \mathbf{q}_k} \left(\frac{\mathbf{w}_i}{\|\mathbf{w}_i\|_2} \right) = \left[\frac{I_D}{\|\mathbf{w}_i\|_2} - \frac{\mathbf{w}_i \mathbf{w}_i^T}{\|\mathbf{w}_i\|_2^3} \right] \frac{\partial \mathbf{w}_i}{\partial \mathbf{q}_k}. \tag{8}$$

Here, I_D is the D-dimensional identity matrix. If we store all $\partial(\mathbf{w}_i / \|\mathbf{w}_i\|_2) / \partial \mathbf{q}_k$ derivatives in a 3D array of dimensions $[d^2, D, D]$, we can form the $\partial W_1 / \partial q_{ij}$ matrices required by (3) by taking 2D slices from this array [3]. The software for the Gram-Schmidt derivatives can be downloaded independently from the DAS method via [5].

Finally, our (MD) surrogate model in the active subspace, denoted by $\widetilde{G}(\mathbf{y})$, is simply the prediction of the neural network from the DAS layer onward. A schematic of the DAS architecture is shown in Figure S1, and for more information on the DAS method we refer to [3, 2].

S3 Kernel-based active subspaces

In dealing with derivative-free problems, the Deep Active Subspace (DAS) method offers a solution by parametrically and explicitly estimating the active subspace with the use of a neural network layer, which is represented by an orthonormal parameter matrix $W_1(Q)$ (see Eq (1)). In contrast, the kernel-based approach estimates the active subspace non-parametrically by computing the derivatives implicitly using kernel methods [6]. The kernel-based active subspace method used in this study is called gradient-based Kernel Dimension Reduction (gKDR) proposed by [7]. [8] integrated gKDR with a Gaussian Process (GP) to formulate a joint framework aimed at emulating high-dimensional simulations. In the forthcoming discussion, we will briefly review the gKDR methods, and then we discuss the strong connections between the GP-gKDR and DAS methods.

Recall that the active subspace approach aims at finding the subspace $\boldsymbol{y} = U_1^T \boldsymbol{x}$ that best explains the variation of the model output $f(\boldsymbol{x})$. If we denote the model output $f(\boldsymbol{x})$ as a random variable F with domain $\mathcal{F}$, the equivalent representation can be written as;

$$p(F|\boldsymbol{x}) = \tilde{p}(F|\boldsymbol{y} = U_1^T \boldsymbol{x}) \text{ or } F \perp \boldsymbol{x} | \boldsymbol{y} = U_1^T \boldsymbol{x}, \tag{9}$$

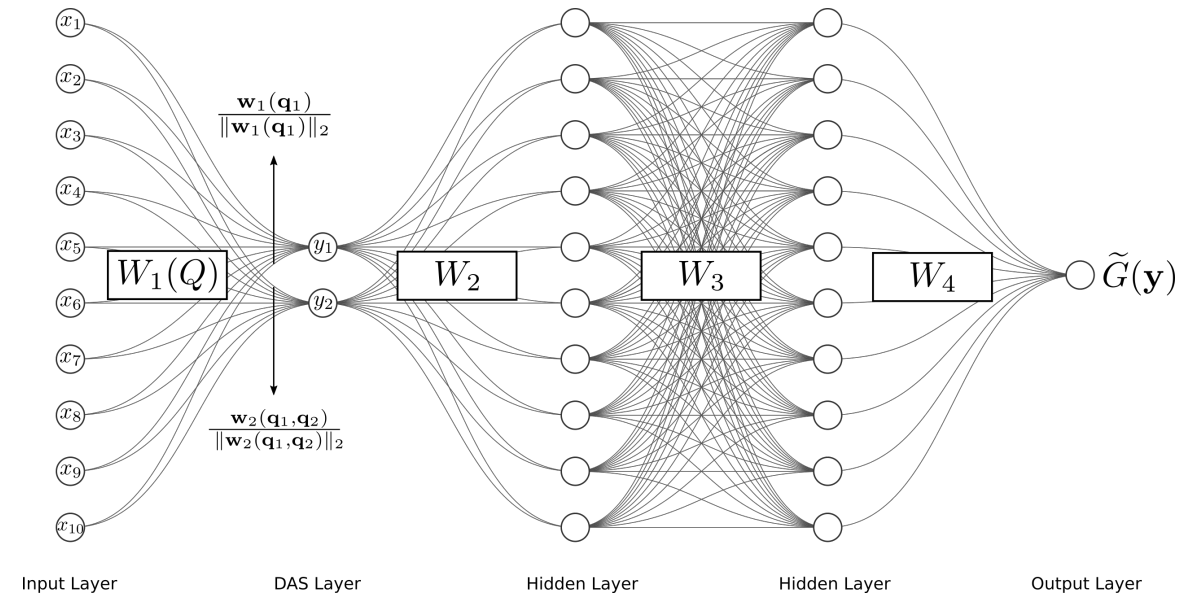


Figure S1: A schematic of the DAS architecture, with a 10-dimensional input space and a two-dimensional DAS layer. The two orthonormal weight vectors that make up $W_1(Q)$ are shown separately. The other weight matrices W_i, $i > 1$ are standard non-orthonormal neural-network weights.

which means that the active variables $\boldsymbol{y} = U_1^T \boldsymbol{x}$ are sufficient to describe the model output F (i.e. the inactive variables can be ignored).

The kernel method is a technique in statistical machine learning that applies a positive-definite (pd) kernel function to implicitly map the input data into a Hilbert space $\mathcal{H}$. A pd kernel function, defined as $k(\cdot,\cdot) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}+$, is uniquely associated to a Hilbert space $\mathcal{H}$, consisting of functions on $\mathbb{R}^D$ such that (i) $k(\boldsymbol{x},\cdot) \in \mathcal{H}$; (ii) the set $\{k(\boldsymbol{x},\cdot)|\boldsymbol{x} \in \mathbb{R}^D\}$ span $\mathcal{H}$, and (iii) $\langle f, k(\boldsymbol{x},\cdot)\rangle_{\mathcal{H}} = f(\boldsymbol{x})$ and $f \in \mathcal{H}$ where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product in $\mathcal{H}$. Property (iii) implies that when the kernel k can recover any function $f \in \mathcal{H}$, the Hilbert space $\mathcal{H}$ is called the Reproducing Kernel Hilbert Space (RKHS) associated with k [6]. Let $k_{\mathcal{X}}$ and $k_{\mathcal{F}}$ be pd kernels on $\mathbb{R}^D$ and $\mathcal{F}$. In the work of Fukumizu and Leng [7], it is shown that, for any $g \in \mathcal{H}_{\mathcal{F}}$, there exists a function $\phi_g : \mathbb{R}^d \to \mathbb{R}$ such that,

$$\mathbb{E}[g(F) \,|\, \boldsymbol{x}] = \phi_g(U_1^T \boldsymbol{x}). \tag{10}$$

The function g is identity function id. Then considering the conditional independence between $f(\boldsymbol{x})$ and $\boldsymbol{x}$ in (9), we recover the conditional expectation over the inactive variables as follows;

$$\mathbb{E}[id(F) \,|\, \boldsymbol{x}] = \mathbb{E}[f(\boldsymbol{x}) \,|\, \boldsymbol{x}] = \mathbb{E}_{\boldsymbol{z}}[f(\boldsymbol{x}) \,|\, \boldsymbol{y} = U_1^T \boldsymbol{x}]. \tag{11}$$

Note that $id \in \mathcal{H}_{\mathcal{F}}$ can be easily verified. Equation (10) thus denotes a generalization applicable to dimension reduction, robust to any nonlinear transformation $g \in \mathcal{H}_{\mathcal{F}}$ on the model output. Under mild assumptions[1], it is established that for any $\boldsymbol{x} \in \mathbb{R}^D$, the following is true for any $g \in \mathcal{H}_F$,

$$\frac{\partial}{\partial x_i}\mathbb{E}[g(F) \,|\, \boldsymbol{x}] = \sum_{j=1}^{d} U_1^{ij} \langle g, \nabla_j \phi(U_1^T \boldsymbol{x})\rangle_{\mathcal{H}_{\mathcal{F}}}, \tag{12}$$

where U_1^{ij} is the (i,j) entry of U_1. Further, let us define the cross-covariance operator $C_{FX} : \mathcal{H}_{\mathcal{X}} \to \mathcal{H}_{\mathcal{F}}$ as the operator such that $\langle h_2, C_{FX} h_1\rangle_{\mathcal{H}_{\mathcal{F}}}$ for any $h_1 \in \mathcal{H}_{\mathcal{X}}, h_2 \in \mathcal{H}_{\mathcal{F}}$. Using the fact that

$$C_{XX}\mathbb{E}[g(F)|\boldsymbol{x}] = C_{XF} g \tag{13}$$

[1]The function $\phi_g(\boldsymbol{y})$ in Eq.(10) is differentiable with respect to $\boldsymbol{y}$, and the linear functional: $g \mapsto \frac{\partial \phi_g(\boldsymbol{y})}{\partial y^i}$ is continuous over for any $\boldsymbol{y} \in \mathbb{R}^M$ where $M \ll D$ and $i = 1, ..., M$.

we establish that, if $\mathbb{E}[g(F) \,|\, \boldsymbol{x}] \in \mathcal{H}_{\mathcal{X}}$, it is feasible to deduce [7]

$$\frac{\partial}{\partial x_i}\mathbb{E}[g(F) \,|\, \boldsymbol{x}] = \langle g \,,\, C_{FX}C_{XX}^{-1}\frac{\partial}{\partial x_i}k_{\mathcal{X}}(\boldsymbol{x} \,,\, \cdot)\rangle_{\mathcal{H}_{\mathcal{F}}}. \tag{14}$$

Equating the Eq (12) and Eq (14) yields, for $i, j = 1, ..., m$,

$$\begin{aligned} M_{ij}(\boldsymbol{x}) &= \langle C_{FX}C_{XX}^{-1}\frac{\partial}{\partial x_i}k_{\mathcal{X}}(\boldsymbol{x} \,,\, \cdot) \,,\, C_{FX}C_{XX}^{-1}\frac{\partial}{\partial x_j}k_{\mathcal{X}}(\boldsymbol{x} \,,\, \cdot)\rangle_{\mathcal{H}_{\mathcal{F}}} \\ &= \sum_{t,s=1}^{d} U_1^{it}U_1^{js}\langle \nabla_t \phi(U_1^T\boldsymbol{x}), \nabla_s \phi(U_1^T\boldsymbol{x})\rangle_{\mathcal{H}_{\mathcal{F}}}. \end{aligned} \tag{15}$$

The dimension reduction projection matrix $U_1 \in \mathbb{R}^{D\times d}$ is consequently formed as the d eigenvectors associated with the largest d eigenvalues of the $D \times D$ matrix $M(\boldsymbol{x})$. In practice, given samples $\mathcal{D} = \{\boldsymbol{x}_i, f(\boldsymbol{x}_i)\}_{i=1}^n$, the matrix M can be estimated with $\tilde{M}$ such that,

$$\tilde{M}_n = \frac{1}{n}\sum_{i=1}^{n} \nabla k_{\mathcal{X}}(\boldsymbol{x}_i)^T(G_{\mathcal{X}} + n\epsilon I)^{-1}G_{\mathcal{F}}(G_{\mathcal{X}} + n\epsilon I)^{-1}\nabla k_{\mathcal{X}}(\boldsymbol{x}_i), \tag{16}$$

where $G_{\mathcal{X}}$ and $G_{\mathcal{F}}$ are the Gram matrices with the (i, j) entry as $k_{\mathcal{X}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $k_{\mathcal{F}}(f(\boldsymbol{x}_i), f(\boldsymbol{x}_j))$ respectively. The $\nabla k_{\mathcal{X}}(\,\cdot\,) = (\partial k_{\mathcal{X}}(\boldsymbol{x}_1 \,,\, \cdot)/\partial \boldsymbol{x}, ..., \partial k_{\mathcal{X}}(\boldsymbol{x}_n \,,\, \cdot)/\partial \boldsymbol{x})^T \in \mathbb{R}^{n\times D}$ and ϵ is the regularization for numerical stability. Subsequently, the estimated $\tilde{U}_1$ is obtained by selecting the d eigenvectors that correspond to the largest d eigenvalues,

$$\tilde{M} = [\tilde{U}_1, \tilde{U}_2]\Lambda(\boldsymbol{x})[\tilde{U}_1, \tilde{U}_2]^T, \Lambda(\boldsymbol{x}) = diag(\lambda_1(\boldsymbol{x}), ..., \lambda_D(\boldsymbol{x})) \tag{17}$$

Furthermore, Gaussian Process Regression (GPR) [9] is performed on the estimated active subspace $\boldsymbol{y} = U_1^T\boldsymbol{x}$ and model output $f(\boldsymbol{x})$. The following procedure is applied to emulate a high-dimensional simulator:

- Step 1. Estimate the projection matrix $\tilde{M}$ by using Eq (16), with a subset of $\mathcal{D}$ containing n_1 model runs.
- Step 2. Obtain the estimated active subspace projection matrix $\tilde{U}_1$ using Eq (17).
- Step 3. Construct the GPR emulator with the lower dimensional representation of whole data set $\tilde{\mathcal{D}} = \{\boldsymbol{y}_i = \tilde{U}_1^T\boldsymbol{x}_i \,,\, f(\boldsymbol{x}_i)\}_{i=1}^n$

Mirroring the GP-gKDR to the DAS method, the gKDR method functions as the DAS layer while the GPR takes on the same role as the subsequent neural network, i.e. the role of the surrogate model. Both frameworks aim at estimating an orthonormal matrix U_1 that projects the original high-dimensional data into a lower-dimensional active subspace, thereby constructing a surrogate model in this subspace. The DAS method explicitly projects the original data $\boldsymbol{x}$ into a high-dimensional space by utilizing a feed-forward neural network, which can be compared to the gKDR implicitly projecting the data into a high-dimensional space where the inner product is induced by the kernel function. In addition, DAS assures the orthonormality of the projection matrix through Gram-Schmidt orthogonalization, whereas GP-gKDR uses an eigen-decomposition. The relationship between kernels and neural networks has attracted much attention and led to considerable advancements in both theoretical understanding and practical applications. A more thorough discussion on these aspects can be found in [10, 4]. It is also noteworthy to highlight a significant difference in the approaches these methods take: the DAS method estimates the active subspace and the surrogate model simultaneously. Conversely, GP-gKDR follows a divide-and-conquer strategy, initially searching for the active subspace in the induced RKHS, and then constructing the GP surrogate model to approximate the function $\phi_g \in \mathcal{H}_{\mathcal{F}}$ as shown in Equation (10). The review [11] provides a more in-depth exploration of the relationship between RKHS and GPR.

S4 Input parameters

Here we list all parameters including their default values that are to beconsidered uncertain.

In the case of materials applications, we use standard 6-12 Lennard-Jones potential for pairwise interactions, harmonic potentials for bonded interactions and the OPLS force field for dihedral interactions. The force field features two parameters (ending with 1 or 2) for the pairwise (starting with `p` interactions, two parameters (ending with 1 or 2) for the bond interactions (starting with `b`), two parameters (ending with 1 or 2) for the angle interactions (starting with `a`), and three parameters (ending with 1, 2 or 3) for the dihedral interactions (starting with `d`). The number in the middle denotes the unique atom type or combination of atom types for a given form of interaction (see Table S1). Although they remain unchanged (in the materials application), we also provide the main simulation parameters (see Table S2).

p11:	0.01	p12:	3.00	p21:	0.01	p22:	2.50	p31:	0.03
p32:	2.42	p41:	0.03	p42:	2.50	p51:	0.07	p52:	3.50
p61:	0.07	p62:	3.55	p71:	0.14	p72:	2.90	p81:	0.17
p82:	3.12	p91:	0.17	p92:	3.30	b11:	268.00	b12:	1.53
b21:	317.00	b22:	1.51	b31:	320.00	b32:	1.41	b41:	340.00
b42:	1.09	b51:	367.00	b52:	1.08	b61:	382.00	b62:	1.45
b71:	434.00	b72:	1.01	b81:	469.00	b82:	1.40	b91:	481.00
b92:	1.34	b101:	553.00	b102:	0.94	a11:	33.00	a12:	107.80
a21:	35.00	a22:	109.50	a31:	35.00	a32:	120.00	a41:	37.50
a42:	110.70	a51:	40.00	a52:	109.50	a61:	43.60	a62:	106.40
a71:	50.00	a72:	109.50	a81:	50.00	a82:	116.00	a91:	51.80
a92:	107.20	a101:	55.00	a102:	108.50	a111:	56.20	a112:	109.47
a121:	58.35	a122:	112.70	a131:	60.00	a132:	109.50	a141:	63.00
a142:	120.00	a151:	70.00	a152:	120.00	d11:	-7.58	d12:	3.43
d13:	3.20	d21:	-1.01	d22:	-0.71	d23:	0.47	d31:	-0.55
d41:	-0.52	d42:	-2.02	d43:	2.00	d51:	-0.36	d52:	-0.17
d53:	0.49	d61:	-0.19	d62:	-0.42	d63:	0.42	d83:	0.30
d93:	0.35	d103:	0.40	d113:	0.47	d123:	0.56	d132:	7.25
d141:	0.42	d142:	-0.13	d143:	0.69	d151:	0.65	d152:	-0.25
d153:	0.67	d161:	1.30	d162:	-0.05	d163:	0.20	d171:	2.39
d172:	-0.67	d173:	0.55	d181:	8.00				

Table S1: The default force-field parameters values encountered in the epoxy polymer materials case.

units:	real
pair_style:	lj/cut/coul/long 12.0Å 9.0Å
kspace_style:	pppm 0.0001
timestep:	1.0 fs
NVT_temperature:	300.0 K
NVT_temperature_damping:	100.0 K
pressure_sampling_time:	100000.0
strain_amplitude:	0.05

Table S2: The default simulation parameter values encountered in the epoxy polymer materials case (left unchanged).

In the case of biomolecular applications, we use the AMBER forcefield. Bond, angle and dihedral interactions (respectively starting with `b`, `a` and `d`) all feature two parameters: an equilibrium value (ending in `ev`, respectively associated with an equilibrium distance, angle and torsional angle); and a force constant (ending in `fc` equivalent to a stiffness). Pairwise interactions (starting with `p`) also feature two parameters: a van der Waals radius of the atom (ending in `rm`) and an energy representing the depth of the potential well (ending in `ep`). The number in the middle denotes the unique atom type

or combination of atom types for a given interaction type (Tables S4 and S6). The default simulation parameters are listed in Tables S3 and S5 for ESMACS and TIES, respectively.

box_size:	14 Å
cutoff:	12 Å
timestep:	2 fs
rigidtolerance:	1e-05 Å
PMEGridSpacing:	1 Å
initTemperature_eq1:	50 K
reassignIncr_eq1:	1
langevinDamping:	5 ps^{-1}
BerendsenPressureCompressibility:	4.57e-05 bar^{-1}
BerendsenPressureRelaxationTime:	100 fs
setTemperature:	300 K
time_factor_eq:	120 ps
BerendsenPressureTarget:	1.01325 bar
time_sim1:	10 ns

Table S3: The default simulation parameter values encountered within the ESMACS absolute free energy calculations.

b01fc:	345.80	b01ev:	1.09	b02fc:	461.10	b02ev:	1.40	b03fc:	637.70
b03ev:	1.22	b04fc:	345.90	b04ev:	1.49	b05fc:	450.20	b05ev:	1.41
b06fc:	351.40	b06ev:	1.49	b07fc:	330.60	b07ev:	1.10	b08fc:	313.00
b08ev:	1.52	b09fc:	427.60	b09ev:	1.38	b10fc:	384.20	b10ev:	1.41
b11fc:	300.90	b11ev:	1.54	b12fc:	330.60	b12ev:	1.10	b13fc:	328.70
b13ev:	1.46	b14fc:	321.00	b14ev:	1.52	b15fc:	404.60	b15ev:	1.01
b16fc:	326.60	b16ev:	1.46	b17fc:	417.90	b17ev:	1.39	b18fc:	305.60
b18ev:	1.75	a01fc:	48.20	a01ev:	2.09	a02fc:	77.90	a02ev:	2.27
a03fc:	64.30	a03ev:	2.10	a04fc:	68.70	a04ev:	2.14	a05fc:	66.60
a05ev:	2.09	a06fc:	66.70	a06ev:	2.07	a07fc:	48.00	a07ev:	2.09
a08fc:	66.30	a08ev:	2.11	a09fc:	64.00	a09ev:	2.11	a10fc:	39.40
a10ev:	1.88	a11fc:	67.40	a11ev:	2.15	a12fc:	63.80	a12ev:	2.16
a13fc:	46.90	a13ev:	1.90	a14fc:	66.80	a14ev:	2.01	a15fc:	74.20
a15ev:	2.15	a16fc:	67.90	a16ev:	2.10	a17fc:	49.80	a17ev:	1.90
a18fc:	46.40	a18ev:	1.91	a19fc:	65.90	a19ev:	1.95	a20fc:	46.30
a20ev:	1.92	a21fc:	63.40	a21ev:	2.11	a22fc:	63.00	a22ev:	2.09
a23fc:	63.10	a23ev:	1.96	a24fc:	62.90	a24ev:	1.95	a25fc:	47.00
a25ev:	1.91	a26fc:	63.50	a26ev:	2.11	a27fc:	46.10	a27ev:	2.02
a28fc:	49.60	a28ev:	1.92	a29fc:	66.20	a29ev:	1.93	a30fc:	66.50
a30ev:	1.94	a31fc:	48.40	a31ev:	2.03	a32fc:	63.40	a32ev:	2.09
a33fc:	68.30	a33ev:	2.11	a34fc:	57.90	a34ev:	2.08	d01fc:	0.75
d02fc:	0.25	d03fc:	0.09	d04fc:	0.61	d05fc:	0.25	d06fc:	0.03
d07fc:	0.15	d08fc:	0.49	d09fc:	0.03	d10fc:	1.00	d11fc:	0.45
d13fc:	0.53	d14fc:	0.15	d15fc:	0.50	d16fc:	1.50	d17fc:	1.05
d18fc:	10.50	d19fc:	1.10	d20fc:	1.00	p01rm:	1.82	p01ep:	0.17
p02rm:	0.60	p02ep:	0.02	p03rm:	1.91	p03ep:	0.11	p04rm:	1.10
p04ep:	0.02	p05rm:	1.39	p05ep:	0.02	p06rm:	1.72	p06ep:	0.21
p08rm:	1.91	p08ep:	0.09	p09rm:	1.66	p09ep:	0.21	p10rm:	1.49
p10ep:	0.02	p11rm:	2.00	p11ep:	0.25	p12rm:	1.46	p12ep:	0.01
p13rm:	1.41	p13ep:	0.01	p14rm:	1.36	p14ep:	0.01	p15rm:	1.95
p15ep:	0.27	p16rm:	1.77	p16ep:	0.15				

Table S4: The default force-field parameter values encountered within the ESMACS absolute free energy calculations.

box_size:	14 Å
cutoff:	12 Å
timestep:	2 fs
rigidtolerance:	1e-05 Å
PMEGridSpacing:	1 Å
langevinDamping:	5 ps^{-1}
BerendsenPressureCompressibility:	4.57e-05 bar^{-1}
BerendsenPressureRelaxationTime:	100 fs
setTemperature:	300 K
time_factor_eq:	40 ps
BerendsenPressureTarget:	1.01325 bar
time_sim1:	4 ns

Table S5: Default simulation parameter values in the TIES case.

b01fc:	375.90	b01ev:	1.10	b02fc:	232.50	b02ev:	1.54	b03fc:	250.30
b03ev:	1.52	b04fc:	395.70	b04ev:	1.09	b05fc:	378.60	b05ev:	1.40
b06fc:	535.10	b06ev:	1.01	b07fc:	349.50	b07ev:	1.38	b08fc:	652.60
b08ev:	1.22	b09fc:	295.40	b09ev:	1.47	b10fc:	354.50	b10ev:	1.38
b11fc:	416.10	b11ev:	1.37	b12fc:	308.20	b12ev:	1.46	b13fc:	262.60
b13ev:	1.50	b14fc:	375.90	b14ev:	1.10	b15fc:	284.80	b15ev:	1.43
b16fc:	357.50	b16ev:	1.37	a01fc:	39.00	a01ev:	1.88	a02fc:	46.80
a02ev:	1.92	a03fc:	65.60	a03ev:	2.11	a04fc:	47.30	a04ev:	1.93
a05fc:	65.20	a05ev:	1.96	a06fc:	64.90	a06ev:	1.95	a07fc:	48.70
a07ev:	2.09	a08fc:	68.80	a08ev:	2.09	a09fc:	47.00	a09ev:	2.19
a10fc:	87.20	a10ev:	2.07	a11fc:	118.80	a11ev:	2.27	a12fc:	68.70
a12ev:	2.15	a13fc:	69.80	a13ev:	1.98	a14fc:	86.70	a14ev:	2.16
a15fc:	47.10	a15ev:	2.19	a16fc:	69.40	a16ev:	1.97	a17fc:	67.20
a17ev:	2.12	a18fc:	92.70	a18ev:	1.87	a19fc:	67.10	a19ev:	2.11
a20fc:	47.70	a20ev:	1.93	a21fc:	66.80	a21ev:	2.08	a22fc:	63.30
a22ev:	2.21	a23fc:	65.50	a23ev:	1.95	a24fc:	46.90	a24ev:	1.91
a25fc:	38.80	a25ev:	1.89	a26fc:	62.40	a26ev:	1.92	a27fc:	85.30
a27ev:	1.88	a28fc:	66.10	a28ev:	2.06	a29fc:	87.30	a29ev:	2.08
a30ev:	2.09	d01fc:	0.12	d02fc:	0.08	d03fc:	0.24	d05fc:	0.16
d06fc:	3.62	d07fc:	0.30	d08fc:	4.00	d09fc:	2.88	d10fc:	1.70
d11fc:	0.70	d12fc:	0.60	d14fc:	0.16	d15fc:	0.25	d16fc:	0.38
d17fc:	1.61	d19fc:	1.10	p01rm:	1.82	p01ep:	0.17	p02rm:	0.60
p02ep:	0.02	p03rm:	1.91	p03ep:	0.11	p04rm:	1.10	p04ep:	0.02
p05rm:	1.91	p05ep:	0.09	p06rm:	1.66	p06ep:	0.21	p07rm:	1.39
p07ep:	0.02	p08rm:	1.49	p08ep:	0.02	p09rm:	1.46	p09ep:	0.01
p10rm:	1.72	p10ep:	0.21	p12rm:	2.00	p12ep:	0.25	p13rm:	1.36
p13ep:	0.01	p14rm:	1.41	p14ep:	0.01	p15rm:	1.86	p15ep:	0.10
p16rm:	1.47	p16ep:	0.02	p17rm:	1.77	p17ep:	0.07	p18rm:	1.91
p18ep:	0.11	p19rm:	1.36	p19ep:	0.02	p20rm:	1.46	p20ep:	0.02
p21rm:	1.71	p21ep:	0.15	p22rm:	1.80	p22ep:	0.20	p23rm:	0.62
p23ep:	0.01	p24rm:	2.25	p24ep:	0.60	p25rm:	1.77	p25ep:	0.15
p26rm:	1.48	p26ep:	0.03						

Table S6: The default force-field parameter values encountered within the TIES relative free energy calculations.

number of hidden layers	2, 3, 4
number of neurons	10, 20, 30, 40
L_2 regularization parameter λ (ANN only)	0.001, 0.01, 0.1

Table S7: The grid-search parameters used in the hyper parameter tuning of the DAS networks.

S5 Hyper parameter tuning and data requirements

For the number of hidden layers after the DAS layer, and the number of neurons per hidden layer we employ a standard grid-search technique [4], using the values shown in Table S7. For each possible combination of hyperparameters, we reserve the last 10% of the data for testing the performance of the network, defined as the average relative L_2 error e_{test}:

$$e_{test} := \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \frac{\|\widetilde{f}(\mathbf{x}_{test,i}) - f(\mathbf{x}_{test,i})\|_2}{\|f(\mathbf{x}_{test,i})\|_2}. \tag{18}$$

Here, $\widetilde{f}$ is the DAS approximation of the MD output f, n_{test} are the number of test data points, $\mathbf{x}_{test,i}$ are the input values of the test data, and the training error e_{train} is defined likewise.

For all networks (and all applications), the standard squared loss function, tanh activation functions, and a minibatch size of 32 is used. Back propagation is performed with stochastic gradient descent using the RMSProp optimizer and a learning rate of 0.001. The number of epochs is automatically determined by invoking early stopping [4], where we terminate training when the relative testing error (18) does not decrease by more than 0.001 over each of the last 3 epochs. To make sure that all inputs have a common scale, we normalize the inputs such that $x_i \in [-1, 1]$ for all $i = 1, \cdots, D$, which is common in the uncertainty-quantification literature.

While our neural networks are small (see Table S7), the grid-search procedure generates ensemble of 2400 neural networks to train, which we offload to the EPCC Archer2 supercomputer at the University of Edinburgh. The ensemble is locally generated by the EasyVVUQ forward-uncertainty propagation toolkit[1], while ensemble execution including data transfer to / from Archer2 is handled via the FabSim3 automation toolkit [12]. FabSim3 is used in conjunction with the QCG-PilotJob framework, which packages all 2400 job into a single allocation, circumventing the limit on the maximum number of jobs (16) that can run on Archer2 at a given time.

For the epoxy application, we are left with $n_{train} = 450$ training data records. To examine the convergence of the errors with data size, we increase the training data size from 50 to 450 in 10 steps, always keeping the size of the test data fixed at 10% of the total data set. For every combination of hyperparameters, and every training data size, we train 20 replica networks to obtain $\overline{e}_{train}$ and $\overline{e}_{test}$, the sample mean of Equation (18) over the replica networks.

The ensemble results are visualized in Figure S2. Here we compare the performance of the DAS network with a standard ANN, and a standard Artificial Neural Network (ANN) with L_2 regularization, where the loss function is modified with a penalty term $\lambda/2\|W\|_2^2$ to promote weight sparsity [4]. Values considered for the hyper parameter λ are also shown in Table S7. The left and middle plots show $\overline{e}_{train}$ and $\overline{e}_{test}$ respectively, and hence show the error distribution due to varying hyper parameters, averaged over the replica networks. For all network types there is a large jump from the training to test error (indicating overfitting), which decreases with more training data. The right plot shows probability distributions of the final test error e_{test}, i.e. using all 450 training data records. The L_2 regularization improves the test performance compared to the ANN without regularization. However, the DAS distribution is even more skewed towards lower test errors. Hence the DAS layer also acts as a form of regularization[3], which outperforms L_2 regularization here.

The ESMACS results are shown in Figure S2b, which are qualitatively similar to the epoxy results, in the sense that the final distribution of the DAS network ($n_{train} = 662$) is more skewed towards lower test errors compared to the ANN with / without L_2 regularization.

Finally, Figure S2c shows the errors for the TIES application. The errors are larger compared to the previous applications. Also, in the TIES application the final DAS distribution ($n_{train} = 439$) is a more significant improvement from the ANN with and without L_2 regularization. While the DAS network once again outperforms the ANNs, it should be noted that the DAS network is slightly harder to train.

Application	n_layers	n_neurons	final average test error $\overline{e}_{test}$
Epoxy	3	10	0.0808
	3	20	0.0810
	3	40	0.0827
ESMACS	2	20	0.0973
	3	10	0.0976
	2	30	0.0985
TIES	3	10	0.24714
	4	10	0.26962
	2	40	0.27043
Epoxy	4	20	0.1148
	4	30	0.1206
	4	40	0.1506
ESMACS	3	30	0.1081
	4	40	0.1089
	4	30	0.1098
TIES	3	30	0.34973
	3	40	0.37623
	4	30	0.41111

Table S8: Grid-search results obtained for the DAS networks, showing the 3 best (top) and 3 worst (bottom) hyper-parameter combinations.

This can be seen by the bump in the left tail of the DAS distribution in Figure S2c, which is caused by DAS replica networks that did not converge properly, i.e. where early stopping cut the training short due to a significant increase in the test error. This happens in the other applications as well, although to a much smaller extent.

Table S8 shows the three best/worst performing hyper-parameter combinations for all network types. Overall, the obtained test ESMACS errors are (slightly) higher than in the epoxy case, despite having access to a larger training set ($n_{train} = 662$ vs $n_{train} = 450$). The TIES errors are significantly higher. For each application we select the indicated best hyper parameter combination for the analysis in the main article.

References

[1] R.A. Richardson, D.W. Wright, W.N. Edeling, V. Jancauskas, J. Lakhlili, and P.V. Coveney. EasyVVUQ: A library for verification, validation and uncertainty quantification in high performance computing. *J. Open Res. Soft.*, 8:11, 2020.

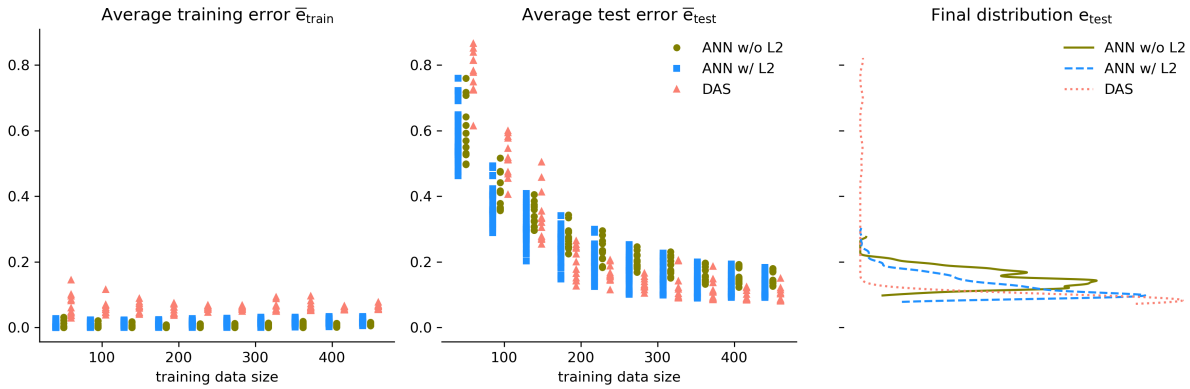
[2] R. Tripathy and I. Bilionis. Deep active subspaces: A scalable method for high-dimensional uncertainty propagation. In *ASME 2019 Int. Des. Eng. Tech. Conf. Comp Inf. Eng. Conf.* American Society of Mechanical Engineers Digital Collection, 2019.

[3] W.N. Edeling. On the deep active-subspace method. *SIAM/ASA J. Uncer. Quant.*, 11(1):62–90, 2023.

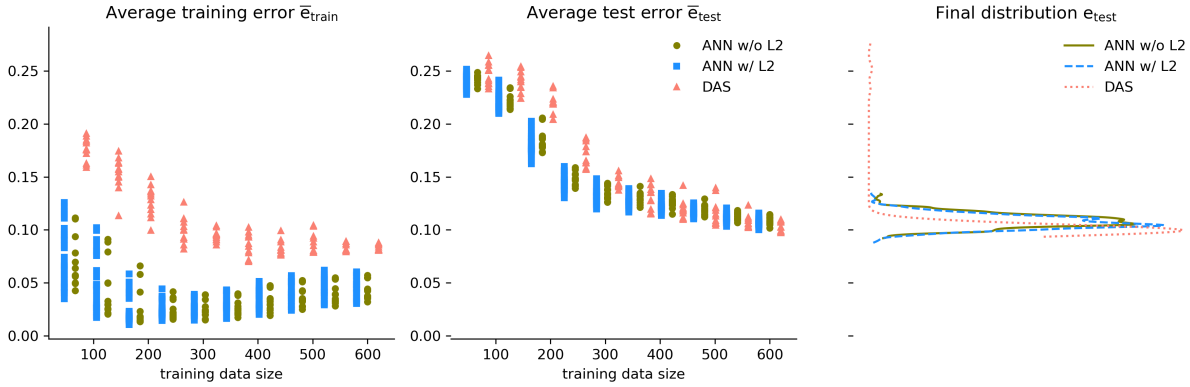
[4] Charu C. Aggarwal. *Neural networks and deep learning: a textbook.* Springer, Cham, Switzerland, 2018.

[5] W.N. Edeling. Gram Schmidt Derivatives (GitHub repository). `https://github.com/wedeling/Gram_Schmidt_Derivatives`, October 2021.

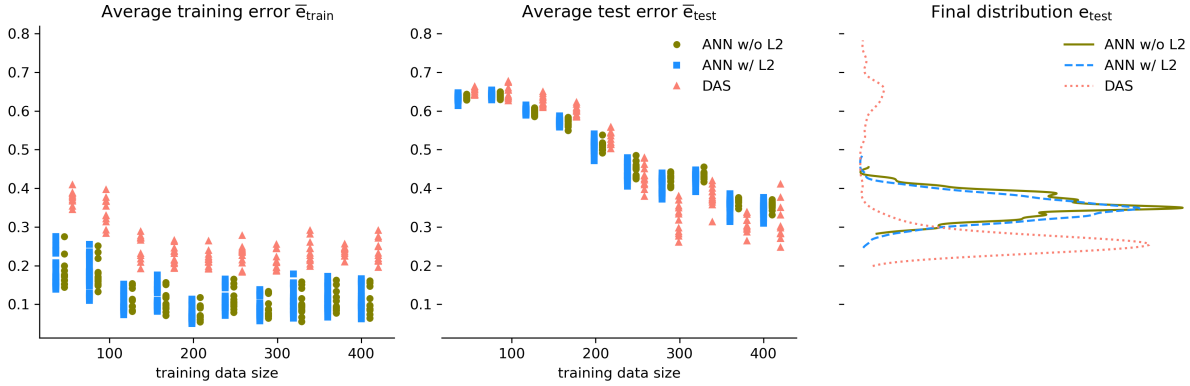
[6] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.



(a) Epoxy application, E_{11} output.



(b) ESMACS application, binding free energy output.



(c) TIES application, relative binding free energy output.

Figure S2: Errors over hyper parameters, replica networks and training data size. Left column: the training error averaged over the replicas for the ANN, ANN with L_2 regularization and the DAS network vs training data size. Middle column: the averaged test error vs training data size. Right column: kernel density estimates for the (non-averaged) final test error, i.e. using all training data.

[7] K. Fukumizu and C. Leng. Gradient-based kernel dimension reduction for regression. *J. Amer. Stat. Assoc.*, 109(505):359–370, 2014.

[8] X. Liu and S. Guillas. Dimension reduction for Gaussian process emulation: An application to the influence of bathymetry on tsunami heights. *SIAM/ASA J. Uncer. Quant.*, 5(1):787–812, 2017.

[9] C.K.I. Williams and C.E. Rasmussen. *Gaussian Processes for Machine Learning*, volume 2. MIT Press, Cambridge, MA, 2006.

[10] J. Lee, Y. Bahri, R. Novak, S.S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[11] M. Kanagawa, P. Hennig, D. Sejdinovic, and B.K. Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences. *arXiv preprint arXiv:1807.02582*, 2018.

[12] D. Groen, H. Arabnejad, D. Suleimenova, W.N. Edeling, E. Raffin, Y. Xue, K. Bronik, N. Monnier, and P.V. Coveney. Fabsim3: An automation toolkit for verified simulations using high performance computing. *Comput. Phys. Comm.*, 283:108596, 2023.