Supporting Information: Machine Learning Approaches toward Orbital-free Density Functional Theory: Simultaneous Training on the Kinetic Energy Density Functional and Its Functional Derivative

Ralf Meyer, Manuel Weichselbaum, and Andreas W. Hauser*

Institute of Experimental Physics, Graz University of Technology, Petersgasse 16, 8010 Graz

E-mail: andreas.w.hauser@gmail.com

In the first section of this Supporting Information the data sets used in the main article are inspected and the accuracy achieved by simple linear models is investigated. Section 2 gives a detailed derivation of the kernel ridge regression algorithm and the method used to include derivative information. The Sections 3 and 4 discuss hyperparameter choices and training procedures for kernel ridge regression and neural networks, respectively. In Section 5 the algorithm used to find minimum energy densities is explained. Section 6 shows the investigations of kernel ridge regression models using a constant offset term, an alternative approach to the principal component analysis method employed in the main article. Section 7 details the influence of the principal component analysis on the functional derivative predictions. Section 8 contains a plot of the learning curve. Section 9 shows computational timings for the various machine learning models.

1 Data Sets

Training and test data are supposed to be created as closely as possible to the data used by Snyder et al.[S1] in order to clearly demonstrate the improved accuracy achieved by including the functional derivative into the training algorithm. In Sections 1.1 and 1.2 the data sets for $N = 1$ and $N = 2$, respectively, are inspected in greater detail. Additionally, the performance achieved by simple models is investigated as reference for the more complex models explored in the main article. The parameter triplets a, b, and c, used to generate the potentials of the training and test set, are available online in CSV format.

1.1 Data for $N = 1$

The training set consists of a total of $M = 100$ densities. Figure S1 shows a typical example of an input density as well as a histogram of the corresponding kinetic energies. In order to evaluate the

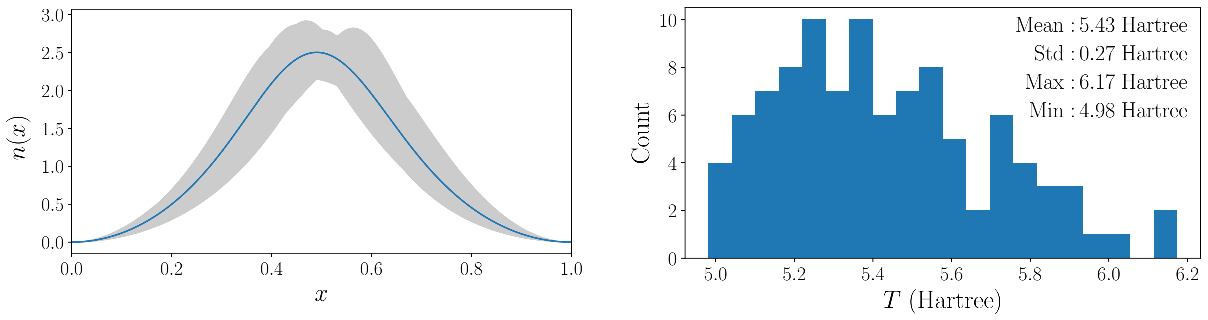

Figure S1: Left panel: The shaded region shows the variation of densities in the training set for $N = 1$. The density corresponding to the first potential in the training set is shown as solid line. Right panel: A Histogram and the statistical parameters of the distribution of kinetic energies in the training set.

complexity of the data set we fit simple regression models and test their performance on the test set. The results for all of these models are summarized in Table S1.

The simplest possible model is a constant model:

$$T^{\text{const}} = b \tag{1}$$

Table S1: Absolute error values for the simple models on the $N = 1$ data set given in kcal/mol.

	$\lvert\Delta T\rvert$		
model	mean	std	max
Constant model	138.3	107.3	847.4
Linear model	2429.8	9924.9	162087.9
Ridge regression	54.3	52.1	417.1

where the least squares solution for the parameter b is given by the mean kinetic energies of the training examples $b = \sum_j^M T_j/M$. The mean absolute error achieved by this model is known as mean absolute deviation (MAD) in statistics. A slightly more complex model is given by a standard least squares linear fit, relating the $G = 500$ dimensional input densities to the kinetic energy:

$$T^{\text{linear}}(\mathbf{n}) = \sum_g^G \mathbf{w}_{(g)}\mathbf{n}_{(g)} = \mathbf{w}^\top \cdot \mathbf{n}, \tag{2}$$

where the index (g) denotes the gth entry of a vector and the parentheses are used to distinguish grid point indices from training example indices. Note that the absolute error for this model is significantly higher than for the constant model. This is most likely due to the fact that 100 training examples are insufficient to fit 500 weight parameters. Ridge regression addresses this problem by introducing an additional regularization term that is used to penalize large weights in the linear fit. Since ridge regression is a linear variant of kernel ridge regression used in the main article, the achieved errors represent an upper bound to the expected performance of KRR. The results are obtained with a regularization parameter of $\lambda = 10^{-6}$ and the ridge regression algorithm as implemented in the scikit-learn python package.[S2]

1.2 Data for $N = 2$

The data set for $N = 2$ used in Section 3.3 and Section 3.4 of the main article is based on the same potentials as the data presented the previous section. While the addition of a second particle leads to higher values for the total kinetic energy, the standard deviation does not increase significantly. We attribute this to the fact that the second particle is less influenced by the potential and the corresponding wave function resembles that of an particle moving freely in a hard wall box. A summary of this training set is given in Figure S2.

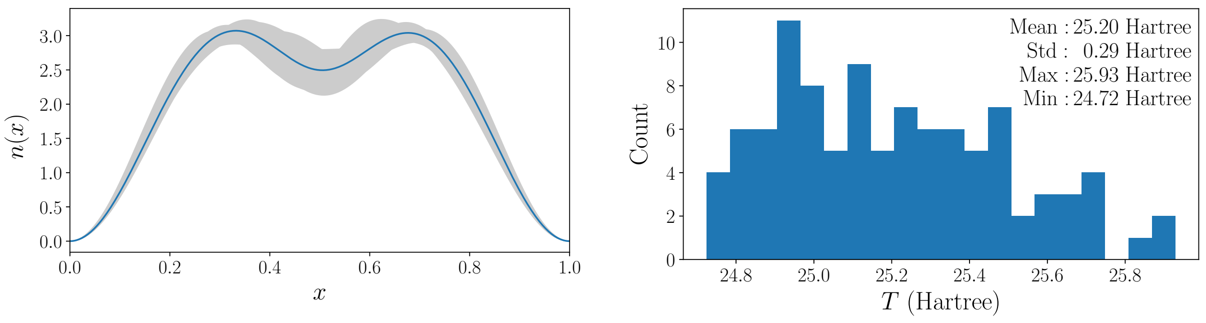

Figure S2: Left panel: The shaded region shows the variation of densities in the training set for $N = 2$. The density corresponding to the first potential in the training set is shown as solid line. Right panel: A Histogram and the statistical parameters of the distribution of kinetic energies.

Table S2 shows the error values achieved by the simple models presented in the previous section. The most notable difference to the results for $N = 1$ is the significant improvement in the accuracy of the linear model.

Table S2: Absolute error values (in kcal/mol) achieved by the simple models on the $N = 2$ test set.

	$\lvert\Delta T\rvert$		
model	mean	std	max
Constant model	145.1	111.7	730.5
Linear model	121.2	409.9	6987.7
Ridge regression	34.6	30.8	322.3

In Section 3.3 and Section 3.4 of the main article the kinetic energy is given by a machine learning correction to the von Weizsäcker functional:

$$T = T^{\mathrm{ML}} + T^{\mathrm{vW}}. \tag{3}$$

The models are therefore not trained on the data set presented in Figure S2 but rather on the difference between these exact values and the prediction obtained with the von Weizsäcker functional.

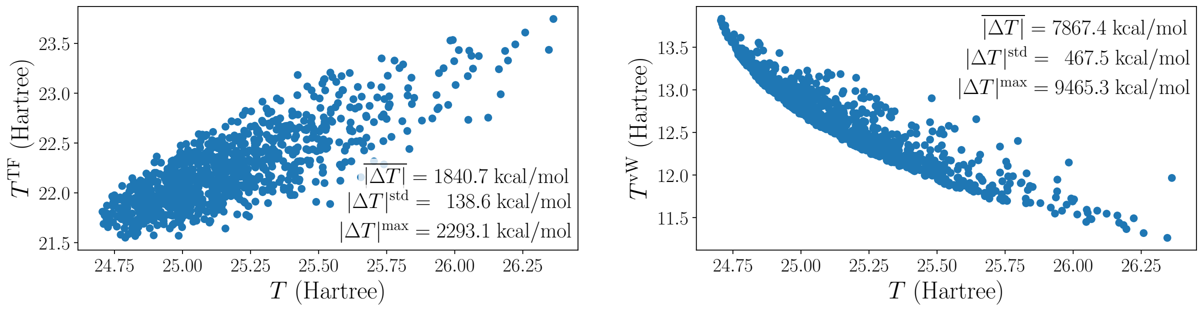

Figure S3: Comparison of the exact kinetic energy values of the $N = 2$ test set on the x-axis and the corresponding predictions by the Thomas-Fermi functional (left panel) and the von Weizsäcker functional (right panel) on the y-axis.

In Figure S3 the kinetic energy predictions of the von Weizsäcker functional and the Thomas-Fermi functional (for the sake of completeness) are plotted versus the exact solutions in order to show that the von Weizsäcker model can not describe systems consisting of two spatial orbitals. While the Thomas-Fermi model roughly captures the correct functional correlation, the predictions by the von Weizsäcker functional exhibit an opposing trend. The accuracy of the Thomas-Fermi model can be improved further by adding a constant term. Fitting this constant offset using the training set yields reduced values of 110.9 kcal/mol, 83.1 kcal/mol, and 453.5 kcal/mol for the mean, the standard deviation and the maximum of the absolute error, respectively.

Due to the opposing functional behavior of the von Weizsäcker model, subtracting the von Weizsäcker kinetic energy from the exact values leads to an increased variance in the training data for the machine learning model as depicted in Figure S4.

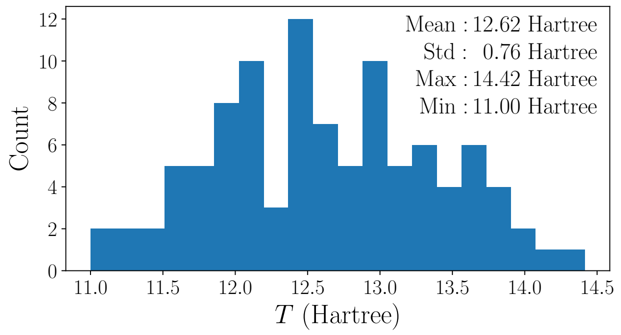

Figure S4: Histogram of the kinetic energies in the training set for $N = 2$ after subtracting the von Weizsäcker kinetic energy.

Table S3 shows that this fact is also reflected in the reduced accuracy achieved by the constant model. Note, however, that both linear models achieve significantly lower mean absolute errors. We conclude that the von Weizsäcker term captures some of the nonlinear contributions to the kinetic energy in this data set.

Table S3: Absolute error values ΔT (in kcal/mol), achieved by the simple models trained on the $N = 2$ data set after subtracting the von Weizsäcker kinetic energy.

	$\lvert\Delta T\rvert$		
model	mean	std	max
Constant model	378.8	279.4	1543.5
Linear model	87.6	413.5	8067.3
Ridge regression	12.9	13.3	113.6

2 Kernel Ridge Regression

In this section we provide a detailed derivation of the Kernel Ridge Regression (KRR) algorithm. In Section 2.1 the KRR method is reviewed and a standard notation valid in both the SI and the main manuscript is introduced. Section 2.2 shows how this concept can be extended to include training data for the functional derivative. Note that in both sections the formulas are derived in the so-called weight space view and transformed only in the last step to the kernel space expressions used in the main article.

2.1 Standard KRR

The main idea in KRR is that even nonlinear data can be described by a linear model after the transformation to a higher dimensional vector space,

$$T^{\mathrm{ML}}(\mathbf{n}) = \sum_{d}^{D} w_d \phi_d(\mathbf{n}) = \mathbf{w}^{\top} \phi(\mathbf{n}), \tag{4}$$

where w_d are the fit coefficients and ϕ denotes the transformation from the input space $\mathbb{R}^G$ to a higher dimensional feature space $\mathbb{R}^D$. The weights are determined by minimization of a cost function consisting of the squared error and a regularization term

$$\mathscr{L} = \sum_{j}^{M} \left(T^{\mathrm{ML}}(\mathbf{n}_j) - T_j\right)^2 + \lambda \sum_{d}^{D} w_d^2, \tag{5}$$

where the parameter λ controls the regularization strength. The cost function is minimized by setting the derivative with respect to w_k equal to zero:

$$\begin{aligned} \frac{\partial \mathscr{L}}{\partial w_k} &= 2\sum_{j}^{M} \left(\sum_{d}^{D} w_d \phi_d(\mathbf{n}_j) - T_j \right) \phi_k(\mathbf{n}_j) + 2\lambda w_k \overset{!}{=} 0 \\ \sum_{j}^{M} \sum_{d}^{D} w_d \phi_d(\mathbf{n}_j)\, \phi_k(\mathbf{n}_j) + \lambda w_k &= \sum_{j}^{M} T_j \phi_k(\mathbf{n}_j). \end{aligned} \tag{6}$$

Derivation with respect to all weights gives a total of D such equations which can be rewritten as a matrix equation:

$$\left(\Phi\Phi^\top + \lambda \mathbf{I}_D\right) \mathbf{w} = \Phi \mathbf{T}, \tag{7}$$

where Φ is a $(D \times M)$ matrix whose columns contain the transformed input densities and $\mathbf{I}_D$ is a unit matrix of size $(D \times D)$. The weights can be calculated by inversion of the matrix on the left hand side:

$$\mathbf{w} = \left(\Phi\Phi^\top + \lambda \mathbf{I}_D\right)^{-1} \Phi \mathbf{T}. \tag{8}$$

This expression for $\mathbf{w}$ can be rearranged further by the following matrix identity, often referred to as push-through identity:[S3–S5]

$$\left(A + P^\top R^{-1} Q\right)^{-1} P^\top R^{-1} = A^{-1} P^\top \left(R + QA^{-1}P^\top\right)^{-1}. \tag{9}$$

Setting $A = \lambda \mathbf{I}_D$, $P = Q = \Phi^\top$ and $R^{-1} = \mathbf{I}_M$ yields:

$$\mathbf{w} = \Phi \left(\Phi^\top \Phi + \lambda \mathbf{I}_M\right)^{-1} \mathbf{T}. \tag{10}$$

One of the advantages of this rearrangement is that the costly matrix inversion can be performed either on the $(D \times D)$ matrix in equation 8 or the $(M \times M)$ matrix in equation 10. As an analog to the D-dimensional weight vector $\mathbf{w}$ the M-dimensional vector $\boldsymbol{\alpha}$ is defined as:

$$\mathbf{w} = \sum_j \alpha_j \phi\left(\mathbf{n}_j\right) \quad \text{with} \quad \boldsymbol{\alpha} = \left(\Phi^\top \Phi + \lambda \mathbf{I}_M\right)^{-1} \mathbf{T}. \tag{11}$$

Plugging the weight vector back into the linear model in equation 4 yields:

$$T^{\mathrm{ML}}(\mathbf{n}) = \mathbf{T} \left(\Phi^\top \Phi + \lambda \mathbf{I}_M\right)^{-1} \Phi^\top \phi(\mathbf{n}) = \boldsymbol{\alpha}^\top \Phi^\top \phi(\mathbf{n}). \tag{12}$$

Another advantage of applying the push-through identity is that it allows for the so-called kernel trick, where the scalar product in feature space is replaced with a kernel function $\phi(\mathbf{n}_i)^\top \phi(\mathbf{n}_j) =$

$k\left(\mathbf{n}_i, \mathbf{n}_j\right)$. After defining the kernel matrix $K = \Phi^\top \Phi$ with elements $K_{ij} = k\left(\mathbf{n}_i, \mathbf{n}_j\right)$ and a vector $\mathbf{k}(\mathbf{n}) = \Phi^\top \phi(\mathbf{n})$ with elements $k_j(\mathbf{n}) = k\left(\mathbf{n}_j, \mathbf{n}\right)$ the final result can be written as:

$$T^{\mathrm{ML}}(\mathbf{n}) = \mathbf{T}\left(K + \lambda \mathbf{I}\right)^{-1} \mathbf{k}(\mathbf{n}) = \boldsymbol{\alpha}^\top \mathbf{k}(\mathbf{n}) = \sum_j^M \alpha_j k\left(\mathbf{n}_j, \mathbf{n}\right). \tag{13}$$

As shown by Snyder et al.[S1] the corresponding prediction for the functional derivative is given by:

$$\frac{\nabla_{\mathbf{n}} T^{\mathrm{ML}}(\mathbf{n})}{\Delta x} = \sum_j^M \frac{\alpha_j}{\Delta x} \nabla_{\mathbf{n}} k\left(\mathbf{n}_j, \mathbf{n}\right). \tag{14}$$

Note that the division by Δx in the discretized functional derivative is necessary to eliminate the dependence on the number of grid points G.

2.2 Including Derivative Information

In a similar fashion, taking the derivative of equation 4 with respect to the input densities, yields the prediction of the functional derivative in weight space:

$$\frac{\nabla_{\mathbf{n}} T^{\mathrm{ML}}(\mathbf{n})}{\Delta x} = \sum_d^D \frac{w_d}{\Delta x} \nabla_{\mathbf{n}} \phi_d(\mathbf{n}). \tag{15}$$

The gth component of the gradient is denoted as:

$$\left(\frac{\nabla_{\mathbf{n}} T^{\mathrm{ML}}(\mathbf{n})}{\Delta x}\right)_{(g)} = \sum_d^D \frac{w_d}{\Delta x} \left(\nabla_{\mathbf{n}} \phi_d(\mathbf{n})\right)_{(g)} = \sum_d^D \frac{w_d}{\Delta x} \frac{\partial \phi_d(\mathbf{n})}{\partial \mathbf{n}_{(g)}}, \tag{16}$$

where the parentheses are used to distinguish grid point indices from training example indices. Using this expression, the cost function can be extended to include the squared error of the functional derivative weighted by an additional hyperparameter κ:

$$\mathscr{L} = \sum_j^M \left(T^{\mathrm{ML}}(\mathbf{n}_j) - T_j\right)^2 + \frac{\kappa}{G} \sum_j^M \sum_g^G \left(\left(\frac{\nabla_{\mathbf{n}_j} T^{\mathrm{ML}}(\mathbf{n}_j)}{\Delta x}\right)_{(g)} - \left(\frac{\nabla_{\mathbf{n}_j} T_j}{\Delta x}\right)_{(g)} \right)^2 + \lambda ||\mathbf{w}||^2, \tag{17}$$

where $\frac{\nabla_{\mathbf{n}_j} T_j}{\Delta x}$ are the reference vectors of the discretized functional derivative. The weights are determined by setting the derivative with respect to w_k equal to zero:

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial w_k} =& 2\sum_j^M \left(\sum_d^D w_d \phi_d\left(\mathbf{n}_j\right) - T_j \right) \phi_k\left(\mathbf{n}_j\right) \\ &+ 2\frac{\kappa}{G}\sum_j^M \sum_g^G \left(\sum_d^D \frac{w_d}{\Delta x} \frac{\partial \phi_d\left(\mathbf{n}_j\right)}{\partial \mathbf{n}_{j,(g)}} - \left(\frac{\nabla_{\mathbf{n}_j} T_j}{\Delta x}\right)_{(g)} \right) \frac{1}{\Delta x} \frac{\partial \phi_k\left(\mathbf{n}_j\right)}{\partial \mathbf{n}_{j,(g)}} + 2\lambda w_k \overset{!}{=} 0.\end{aligned} \tag{18}$$

Collecting all terms proportional to the weights on the left hand side yields

$$\begin{aligned}&\sum_j^M \sum_d^D \left(w_d \phi_d\left(\mathbf{n}_j\right) \phi_k\left(\mathbf{n}_j\right) + \frac{\kappa}{G}\sum_g^G \frac{w_d}{(\Delta x)^2} \frac{\partial \phi_d\left(\mathbf{n}_j\right)}{\partial \mathbf{n}_{j,(g)}} \frac{\partial \phi_k\left(\mathbf{n}_j\right)}{\partial \mathbf{n}_{j,(g)}} \right) + \lambda w_k \\ &= \sum_j^M \left(T_j \phi_k\left(\mathbf{n}_j\right) + \frac{\kappa}{G}\sum_g^G \left(\frac{\nabla_{\mathbf{n}_j} T_j}{\Delta x}\right)_{(g)} \frac{1}{\Delta x} \frac{\partial \phi_k\left(\mathbf{n}_j\right)}{\partial \mathbf{n}_{j,(g)}} \right).\end{aligned} \tag{19}$$

By extending the matrices defined in equation 7 this can again be rewritten as a matrix equation

$$\left(\Phi_{\text{ext}}\, S\, \Phi_{\text{ext}}^{\top} + \lambda \mathbf{I}_D\right) \mathbf{w} = \Phi_{\text{ext}}\, S\, \mathbf{T}_{\text{ext}}, \tag{20}$$

with the extended transformation matrix of shape $(D \times M(1+G))$:

$$\begin{aligned}\Phi_{\text{ext}} &= \begin{pmatrix} \phi_1(\mathbf{n}_1) & \dots & \phi_1(\mathbf{n}_M) & \frac{\nabla_{\mathbf{n}_1}^{\top} \phi_1(\mathbf{n}_1)}{\Delta x} & \dots & \frac{\nabla_{\mathbf{n}_M}^{\top} \phi_1(\mathbf{n}_M)}{\Delta x} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \phi_D(\mathbf{n}_1) & \dots & \phi_D(\mathbf{n}_M) & \frac{\nabla_{\mathbf{n}_1}^{\top} \phi_D(\mathbf{n}_1)}{\Delta x} & \dots & \frac{\nabla_{\mathbf{n}_M}^{\top} \phi_D(\mathbf{n}_M)}{\Delta x} \end{pmatrix} = \\ &= \begin{pmatrix} \phi_1(\mathbf{n}_1) & \dots & \phi_1(\mathbf{n}_M) & \frac{1}{\Delta x}\frac{\partial \phi_1(\mathbf{n}_1)}{\partial \mathbf{n}_{1,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial \phi_1(\mathbf{n}_1)}{\partial \mathbf{n}_{1,(G)}} & \dots & \frac{1}{\Delta x}\frac{\partial \phi_1(\mathbf{n}_M)}{\partial \mathbf{n}_{M,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial \phi_1(\mathbf{n}_M)}{\partial \mathbf{n}_{M,(G)}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \phi_D(\mathbf{n}_1) & \dots & \phi_D(\mathbf{n}_M) & \frac{1}{\Delta x}\frac{\partial \phi_D(\mathbf{n}_1)}{\partial \mathbf{n}_{1,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial \phi_D(\mathbf{n}_1)}{\partial \mathbf{n}_{1,(G)}} & \dots & \frac{1}{\Delta x}\frac{\partial \phi_D(\mathbf{n}_M)}{\partial \mathbf{n}_{M,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial \phi_D(\mathbf{n}_M)}{\partial \mathbf{n}_{M,(G)}} \end{pmatrix},\end{aligned} \tag{21}$$

the extended target vector:

$$\mathbf{T}_{\text{ext}} = \begin{pmatrix} T_1 & \dots & T_M & \frac{\nabla_{\mathbf{n}_1}^\top T_1}{\Delta x} & \dots & \frac{\nabla_{\mathbf{n}_M}^\top T_M}{\Delta x} \end{pmatrix}^\top = \\ = \begin{pmatrix} T_1 & \dots & T_M & \left(\frac{\nabla_{\mathbf{n}_1} T_1}{\Delta x}\right)_{(1)} & \dots & \left(\frac{\nabla_{\mathbf{n}_1} T_1}{\Delta x}\right)_{(G)} & \dots & \left(\frac{\nabla_{\mathbf{n}_M} T_M}{\Delta x}\right)_{(1)} & \dots & \left(\frac{\nabla_{\mathbf{n}_M} T_M}{\Delta x}\right)_{(G)} \end{pmatrix}^\top, \tag{22}$$

and a $(M(1+G) \times M(1+G))$ diagonal matrix containing the scaling factor for the relative importance of derivative information:

$$S = \begin{pmatrix} \mathbf{I}_M & 0 \\ 0 & \frac{\kappa}{G}\mathbf{I}_{MG} \end{pmatrix}. \tag{23}$$

Solving equation 20 for the weight vector $\mathbf{w}$ and applying the push-through identity of equation 9 by setting $A = \lambda \mathbf{I}_D$, $P = Q = \Phi_{\text{ext}}^\top$ and $R^{-1} = S$ yields

$$\mathbf{w} = \left(\Phi_{\text{ext}}\, S\, \Phi_{\text{ext}}^\top + \lambda \mathbf{I}_D\right)^{-1} \Phi_{\text{ext}}\, S\, \mathbf{T}_{\text{ext}} = \Phi_{\text{ext}} \left(\Phi_{\text{ext}}^\top \Phi_{\text{ext}} + \Lambda\right)^{-1} \mathbf{T}_{\text{ext}}, \tag{24}$$

with the regularization matrix

$$\Lambda = \lambda S^{-1} = \begin{pmatrix} \lambda \mathbf{I}_M & 0 \\ 0 & \frac{\lambda G}{\kappa}\mathbf{I}_{MG} \end{pmatrix}. \tag{25}$$

The weights are again rewritten as

$$\mathbf{w}^\top = \sum_j \alpha_j \phi(\mathbf{n}_j)^\top + \frac{\boldsymbol{\beta}_j^\top \cdot \left(\nabla_{\mathbf{n}_j} \phi(\mathbf{n}_j)^\top\right)}{\Delta x}, \tag{26}$$

where the coefficients α and β are determined by solving the matrix equation

$$\begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_M \\ \beta_1 \\ \vdots \\ \beta_M \end{pmatrix} = \left(\Phi_{\text{ext}}^{\top}\Phi_{\text{ext}} + \Lambda\right)^{-1} \begin{pmatrix} T_1 \\ \vdots \\ T_M \\ \frac{\nabla_{\mathbf{n}_1} T_1}{\Delta x} \\ \vdots \\ \frac{\nabla_{\mathbf{n}_M} T_M}{\Delta x} \end{pmatrix} = \left(K_{\text{ext}} + \Lambda\right)^{-1} \mathbf{T}_{\text{ext}}. \tag{27}$$

Note that the individual β_j are vectors of length G. The extended kernel matrix K_{ext} is calculated by applying the kernel trick to the extended transformation matrix Φ_{ext}:

$$K_{\text{ext}} = \begin{pmatrix} K & J' \\ J & H \end{pmatrix} = \begin{pmatrix} k(\mathbf{n}_1,\mathbf{n}_1) & \dots & k(\mathbf{n}_1,\mathbf{n}_M) & \frac{\nabla_{\mathbf{n}_1}^{\top} k(\mathbf{n}_1,\mathbf{n}_1)}{\Delta x} & \dots & \frac{\nabla_{\mathbf{n}_M}^{\top} k(\mathbf{n}_1,\mathbf{n}_M)}{\Delta x} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{n}_M,\mathbf{n}_1) & \dots & k(\mathbf{n}_M,\mathbf{n}_M) & \frac{\nabla_{\mathbf{n}_1}^{\top} k(\mathbf{n}_M,\mathbf{n}_1)}{\Delta x} & \dots & \frac{\nabla_{\mathbf{n}_M}^{\top} k(\mathbf{n}_M,\mathbf{n}_M)}{\Delta x} \\ \frac{\nabla_{\mathbf{n}_1} k(\mathbf{n}_1,\mathbf{n}_1)}{\Delta x} & \dots & \frac{\nabla_{\mathbf{n}_1} k(\mathbf{n}_1,\mathbf{n}_M)}{\Delta x} & \frac{\nabla_{\mathbf{n}_1} \cdot \nabla_{\mathbf{n}_1}^{\top} k(\mathbf{n}_1,\mathbf{n}_1)}{(\Delta x)^2} & \dots & \frac{\nabla_{\mathbf{n}_1} \cdot \nabla_{\mathbf{n}_M}^{\top} k(\mathbf{n}_1,\mathbf{n}_M)}{(\Delta x)^2} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\nabla_{\mathbf{n}_M} k(\mathbf{n}_M,\mathbf{n}_1)}{\Delta x} & \dots & \frac{\nabla_{\mathbf{n}_M} k(\mathbf{n}_M,\mathbf{n}_M)}{\Delta x} & \frac{\nabla_{\mathbf{n}_M} \cdot \nabla_{\mathbf{n}_1}^{\top} k(\mathbf{n}_M,\mathbf{n}_1)}{(\Delta x)^2} & \dots & \frac{\nabla_{\mathbf{n}_M} \cdot \nabla_{\mathbf{n}_M}^{\top} k(\mathbf{n}_M,\mathbf{n}_M)}{(\Delta x)^2} \end{pmatrix}$$

$$= \begin{pmatrix} k(\mathbf{n}_1,\mathbf{n}_1) & \dots & k(\mathbf{n}_1,\mathbf{n}_M) & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_1,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_1,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(G)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_1,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_1,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(G)}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ k(\mathbf{n}_M,\mathbf{n}_1) & \dots & k(\mathbf{n}_M,\mathbf{n}_M) & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_M,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_M,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(G)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_M,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_M,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(G)}} \\ \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_1,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_1,\mathbf{n}_M)}{\partial \mathbf{n}_{1,(1)}} & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_1,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(1)}\partial \mathbf{n}_{1,(1)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_1,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(1)}\partial \mathbf{n}_{1,(G)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_1,\mathbf{n}_M)}{\partial \mathbf{n}_{1,(1)}\partial \mathbf{n}_{M,(1)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_1,\mathbf{n}_M)}{\partial \mathbf{n}_{1,(1)}\partial \mathbf{n}_{M,(G)}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_1,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(G)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_1,\mathbf{n}_M)}{\partial \mathbf{n}_{1,(G)}} & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_1,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(G)}\partial \mathbf{n}_{1,(1)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_1,\mathbf{n}_1)}{\partial \mathbf{n}_{1,(G)}\partial \mathbf{n}_{1,(G)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_1,\mathbf{n}_M)}{\partial \mathbf{n}_{1,(G)}\partial \mathbf{n}_{M,(1)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_1,\mathbf{n}_M)}{\partial \mathbf{n}_{1,(G)}\partial \mathbf{n}_{M,(G)}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_M,\mathbf{n}_1)}{\partial \mathbf{n}_{M,(1)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_M,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(1)}} & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_M,\mathbf{n}_1)}{\partial \mathbf{n}_{M,(1)}\partial \mathbf{n}_{1,(1)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_M,\mathbf{n}_1)}{\partial \mathbf{n}_{M,(1)}\partial \mathbf{n}_{1,(G)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_M,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(1)}\partial \mathbf{n}_{M,(1)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_M,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(1)}\partial \mathbf{n}_{M,(G)}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_M,\mathbf{n}_1)}{\partial \mathbf{n}_{M,(G)}} & \dots & \frac{1}{\Delta x}\frac{\partial k(\mathbf{n}_M,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(G)}} & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_M,\mathbf{n}_1)}{\partial \mathbf{n}_{M,(G)}\partial \mathbf{n}_{1,(1)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_M,\mathbf{n}_1)}{\partial \mathbf{n}_{M,(G)}\partial \mathbf{n}_{1,(G)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_M,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(G)}\partial \mathbf{n}_{M,(1)}} & \dots & \frac{1}{(\Delta x)^2}\frac{\partial^2 k(\mathbf{n}_M,\mathbf{n}_M)}{\partial \mathbf{n}_{M,(G)}\partial \mathbf{n}_{M,(G)}} \end{pmatrix}. \tag{28}$$

Using the new weights from equation 26 for the prediction of the kinetic energy of a previously unseen density **n** in equation 4 yields:

$$T^{\mathrm{ML}}(\mathbf{n}) = \sum_j^M \alpha_j k(\mathbf{n}_j, \mathbf{n}) + \sum_j^M \sum_g^G \frac{\beta_{j,(g)}}{\Delta x} \frac{\partial k(\mathbf{n}_j, \mathbf{n})}{\partial \mathbf{n}_{j,(g)}} = \sum_j^M \alpha_j k(\mathbf{n}_j, \mathbf{n}) + \sum_j^M \frac{\boldsymbol{\beta}_j^\top \cdot \nabla_{\mathbf{n}_j} k(\mathbf{n}_j, \mathbf{n})}{\Delta x}. \quad (29)$$

The corresponding prediction of the derivative is given by:

$$\left(\frac{\nabla_{\mathbf{n}} T^{\mathrm{ML}}(\mathbf{n})}{\Delta x} \right)_{(g)} = \sum_j^M \frac{\alpha_j}{\Delta x} \frac{\partial k(\mathbf{n}_j, \mathbf{n})}{\partial \mathbf{n}_{(g)}} + \sum_j^M \sum_{g'}^G \frac{\boldsymbol{\beta}_{j,(g')}}{(\Delta x)^2} \frac{\partial^2 k(\mathbf{n}_j, \mathbf{n})}{\partial \mathbf{n}_{j,(g')} \partial \mathbf{n}_{(g)}}, \quad (30)$$

or similarly in the vector notation used in the main article by:

$$\frac{\nabla_{\mathbf{n}}^\top T^{\mathrm{ML}}(\mathbf{n})}{\Delta x} = \sum_j^M \frac{\alpha_j}{\Delta x} \nabla_{\mathbf{n}}^\top k(\mathbf{n}_j, \mathbf{n}) + \sum_j^M \frac{\boldsymbol{\beta}_j^\top \cdot \left(\nabla_{\mathbf{n}_j} \cdot \nabla_{\mathbf{n}}^\top k(\mathbf{n}_j, \mathbf{n}) \right)}{(\Delta x)^2}. \quad (31)$$

3 KRR Hyperparameter Search

The hyperparameters for the KRR models are determined using 5-fold cross validation on a rectangular search grid. A total of 29 points, log-uniformly distributed between 10 and 500, are used for the length scale parameter σ, and 5 points, log-uniformly distributed between 10^{-10} and 10^{-14}, for the regularization parameter λ. Both the mean absolute error for the kinetic energy and the functional derivative are evaluated using cross validation. The hyperparameters are chosen such that the sum of these two errors is minimized. Note that this choice is biased toward the derivative score as the mean absolute error on the derivative is typically significantly larger.

As a first test we investigate if the extended KRR model yields similar tendencies for the hyperparameters when the number of training examples is increased as reported for standard KRR in Ref. S1. Figure S5 shows that the rough grid search in fact displays a similar trend toward smaller values for both σ and λ for an increasing training set size M. As expected for a machine

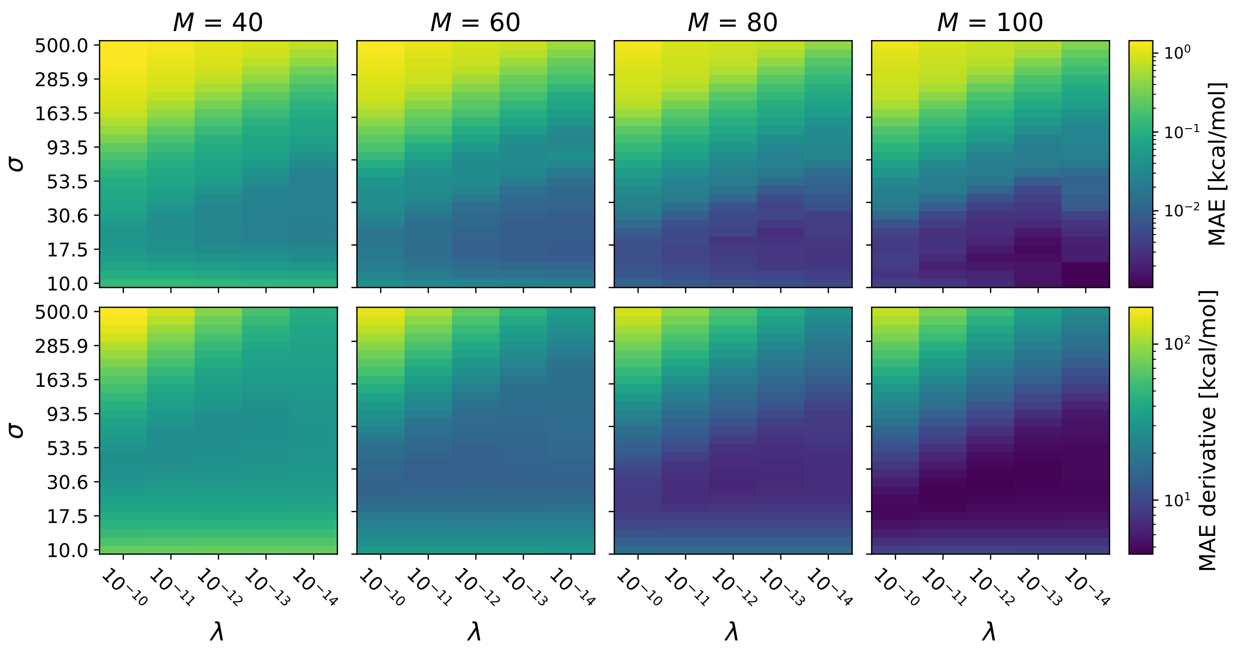

Figure S5: Mean absolute error of the kinetic energy (top row) and the functional derivative (bottom row) as a function of the hyperparameters σ and λ for an increasing number of training examples M and $\kappa = 1$.

learning model, both errors decrease with the number of training examples. Chemical accuracy, defined as a mean absolute error for the kinetic energy below 1 kcal/mol, is already reached using $M = 40$ training examples for a broad range of hyperparameters. This improvement over the $M = 80$ necessary training examples reported by Snyder et al.[S1] is attributed to the inclusion of derivative information. The lowest summed error on the $M = 100$ set is achieved for $\sigma = 30.58$ and $\lambda = 10^{-12}$.

The influence of the weighting parameter κ is investigated by repeating the grid search on $M = 100$ training examples for various values of κ. Figure S6 shows the cross validation score for both the kinetic energy and the functional derivative for $\kappa \in \{0.1, 1, 10, 100, 1000\}$.

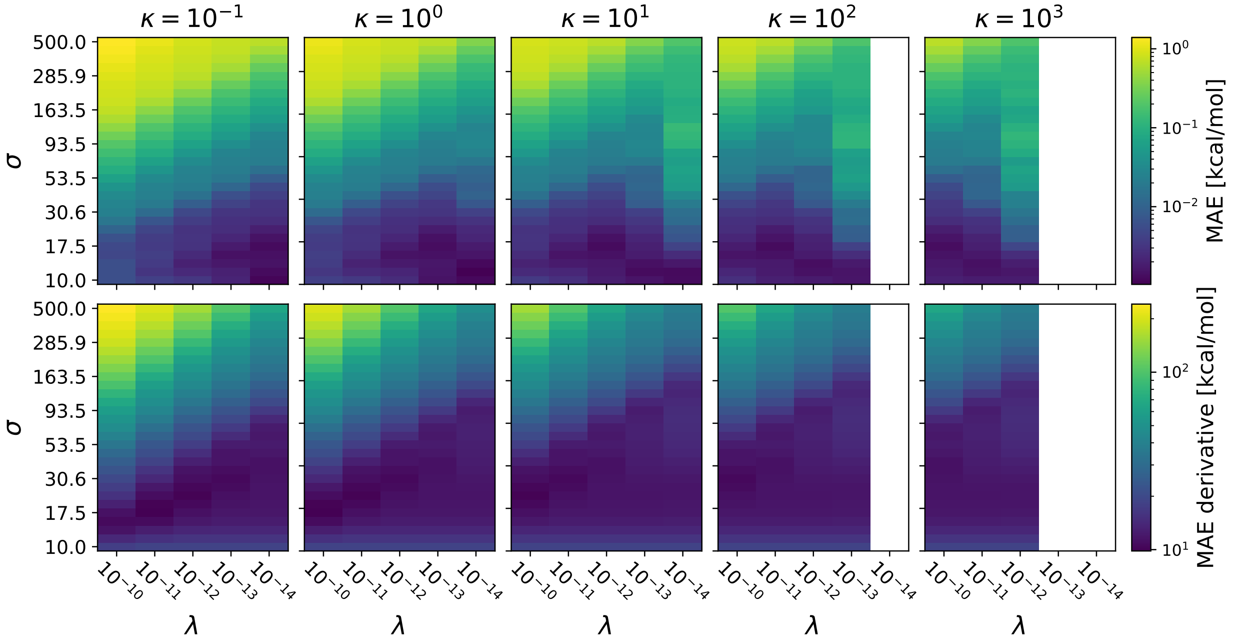

Figure S6: Mean absolute error of the kinetic energy (top row) and the functional derivative (bottom row) for the different values of the hyperparameters κ, σ and λ on the $M = 100$, $N = 1$ training set.

Note that the blank regions in Figure S6 denote areas where the matrix inversion in equation 24, more precisely the Cholesky factorization used to solve equation 27, failed due to numerical noise. As expected, the larger values of κ lead to an increased parameter range with low errors on the functional derivative and conversely a decreased parameter range with the lowest errors for the kinetic energy. Note, however, that the parameter region in which chemical accuracy is reached increases for large κ values. The overall lowest summed error is reached for $\kappa = 0.1$, $\sigma = 17.49$

and $\lambda = 10^{-11}$. Nevertheless, the weighting parameter is set to $\kappa = 1$ for all of the hyperparameter investigations, since it is difficult to judge the relative importance of the errors for the presented application.

Figure S7 shows the results of the hyperparameter search for the $N = 2$ data set. The best performance is achieved for values of $\sigma = 35.16$, $\lambda = 10^{-12}$ and $\sigma = 26.59$, $\lambda = 10^{-12}$ for the standard and extended KRR models, respectively. The hyperparameters for standard KRR are chosen based solely on the MAE of the kinetic energy and a value of $\kappa = 1.0$ is used for extended KRR. Again, the inclusion of derivative information significantly improves the error on the kinetic energy for a wide range of hyperparameters, yielding chemical accuracy on all points of the hyperparameter grid.

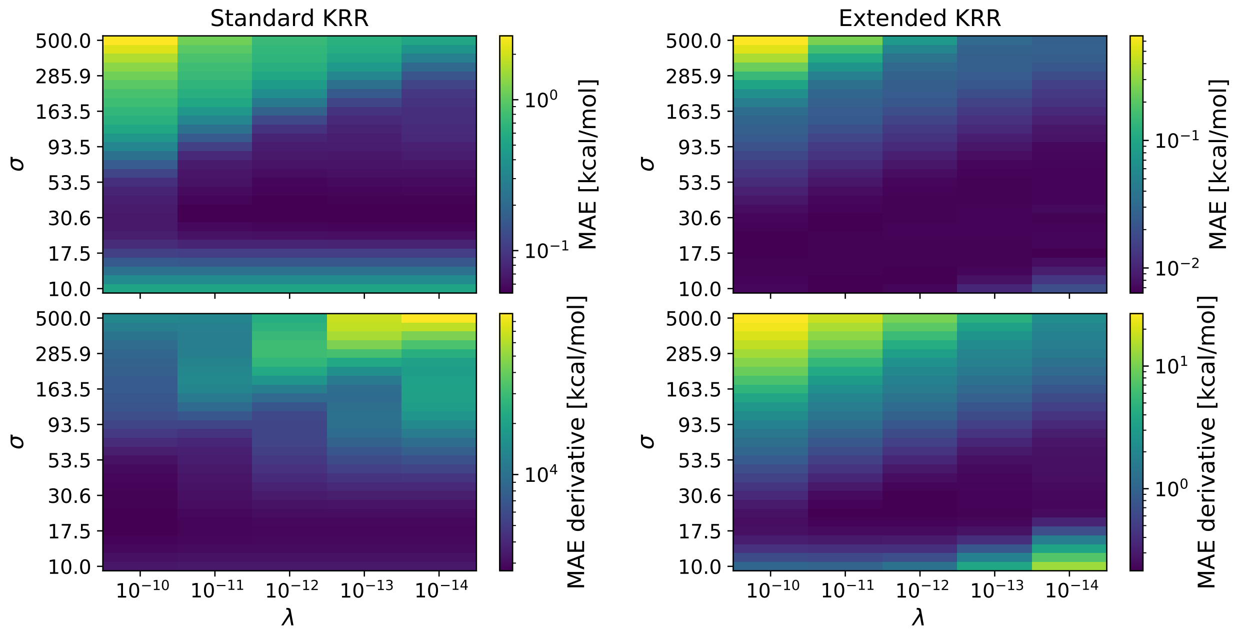

Figure S7: Cross validation scores of the grid search for both the standard KRR model (left) and the extended KRR model (right) on the $N = 2$ training set.

4 Neural Network Hyperparameters and Training Behavior

This section summarizes all the hyperparameters used during the training of the convolutional neural network models for the sake of reproducibility, but without discussing their influence on the actual training behavior or the final model performance.

The neural network models are trained by minimizing the following cost function:

$$\begin{aligned}\mathscr{L} =& \frac{\iota_T}{M}\sum_j^M ||T^{\mathrm{ML}}(\mathbf{n}_j) - T_j||^2 + \frac{\iota_\tau}{MG}\sum_j^M ||\boldsymbol{\tau}^{\mathrm{ML}}(\mathbf{n}_j) - \boldsymbol{\tau}_j||^2 \\ &+ \frac{\kappa}{MG}\sum_j^M \left|\left| \frac{\nabla_{\mathbf{n}_j} T^{\mathrm{ML}}(\mathbf{n}_j)}{\Delta x} - \frac{\nabla_{\mathbf{n}_j} T_j}{\Delta x} \right|\right|^2 + \lambda ||\mathbf{w}||^2.\end{aligned} \tag{32}$$

Most choices for the hyperparameters are shared by all of the models. The remaining hyperparameters are listed in Table S4. Since the relative weighting of contributions in the cost function is

Table S4: Summary of the hyperparameters used for training of the neural networks.

model	N	data set[a]	vW[b]	M	epochs	ι_T	ι_τ	λ	N_{constant}	N_{decay}
CNN	1	recreated	-	100	100 000	0.2	-	0.00025	21 800	1000
ResNet	1	recreated	-	100	100 000	0	1	0.00025	21 800	1000
ResNet	2	recreated	Yes	100	100 000	0	1	0.00025	21 800	1000
ResNet	2	generated	Yes	1000	30 000	0	1	0.000025	40 000	2000
ResNet	2	generated	Yes	10 000	3000	0	1	0.000025	40 000	2000
ResNet	2	generated	Yes	100 000	300	0	1	0.0000001	40 000	2000

[a] *recreated* refers to the fact that the parameters for the potentials are taken from Ref. S1 whereas *generated* refers to new randomly generated parameters.

[b] Denotes that the von Weizsäcker predictions have been subtracted before training.

over determined by four scaling factors, the weighting parameter κ is set to a fixed value of $\kappa = 1$. The network parameters are determined with the Adam optimizer,[S6] a variation of the steepest descent algorithm, with the tensorflow[S7] default hyperparameters of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-7}$. The gradient of the cost function with respect to the weights and biases is calculated using a batch size of 100 examples. This corresponds to the whole training set for the investigations in Section 3.1 through Section 3.3 of the main article. If the norm of the gradient surpasses a threshold value of 100 it is rescaled to a length of 100. This process is referred to as *clip by norm* in tensorflow. Starting from an initial learning rate of 10^{-4} the learning rate stays constant for the first N_{constant} training steps and is then reduced by a factor 0.9 every N_{decay} steps.

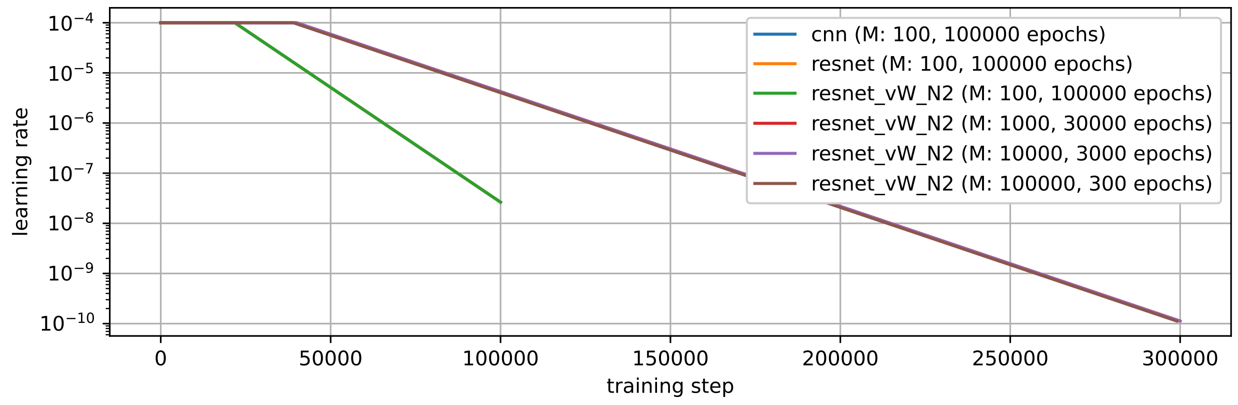

Figure S8: Plot of the learning rate schedule.

Figure S8 shows the learning rate schedule used for training the convolutional neural networks. The number of training steps on the x-axis refers to the number of weight updates and is given by the number of epochs times M divided by the batch size.

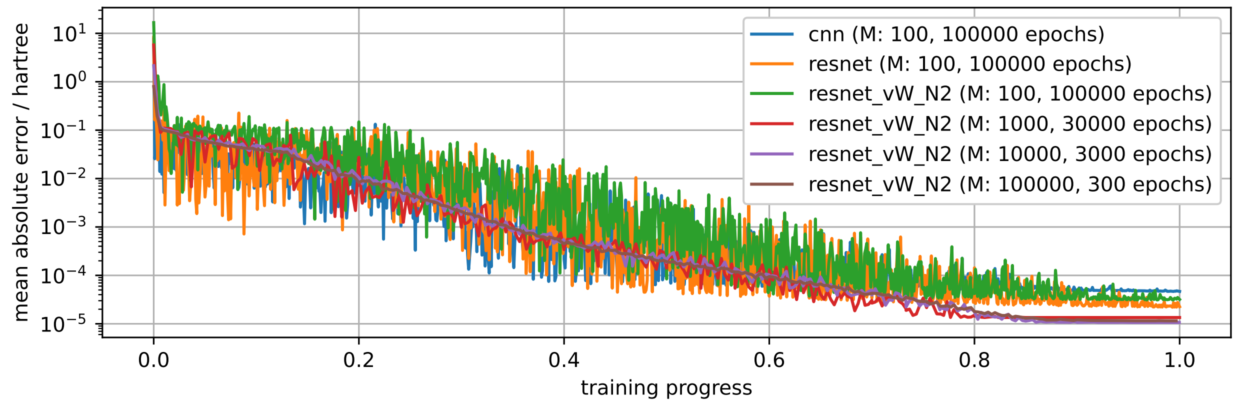

Figure S9: Plot of the mean absolute error of the kinetic energy during the training of the neural network models.

In Figure S9 the mean absolute errors for the kinetic energy on the training set are plotted as a function of the training progress. It shows that in addition to the lower final error reached by ResNets trained on larger data sets, the variation in the error curve during the training is also reduced. For all of the presented models the final training error in Figure S9 is significantly lower than the reported test errors. The generalization properties can be improved by using additional training data as shown in Section 3.4 of the main article.

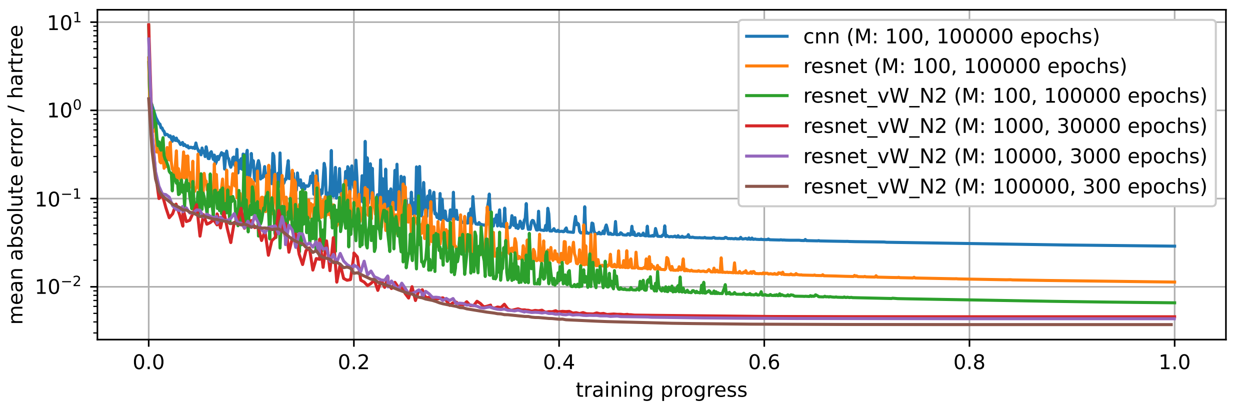

Figure S10: Plot of the mean absolute error of the kinetic energy derivative during the training of the neural network models.

Figure S10 shows the mean absolute error for the functional derivative as a function of the training process. Comparing Figure S9 and S10 shows that the early stages of the training are dominated by a reduction in the derivative error. In part due to the decreasing learning rate, the derivative error stops improving roughly at the half-way point of the training process while the kinetic energy error keeps decreasing.

5 Finding Minimum Energy Densities

5.1 Iteration on the Density and Local Principal Component Analysis

The direct minimization algorithm used to find the minimum energy density is a standard steepest descent optimization using a Lagrange multiplier μ to ensure the correct number of particles N. The corresponding Lagrange function is given by:

$$\mathscr{L}[n] = E[n] - \mu \left(\int n \,\mathrm{d}x - N \right), \tag{33}$$

where the total energy functional $E[n]$ for noninteracting particles is simply the sum of kinetic energy $T[n]$ and potential energy:

$$E[n] = T[n] + \int nV \,\mathrm{d}x. \tag{34}$$

The steepest descent update rule with step size η is given by:

$$n_{i+1} = n_i - \eta \frac{\delta \mathscr{L}}{\delta n} = n_i - \eta \left(\frac{\delta T[n]}{\delta n} + V - \mu \right), \tag{35}$$

where the Lagrange multiplier μ is chosen such that the norm $\int n_{i+1} \,\mathrm{d}x = N$ is ensured. Assuming that n_i is properly normalized, this is equivalent to

$$\int \left(\frac{\delta T[n]}{\delta n} + V - \mu \right) \mathrm{d}x \overset{!}{=} 0 \tag{36}$$

$$\mu = \frac{\int \left(\frac{\delta T}{\delta n} + V \right) \mathrm{d}x}{\int 1 \,\mathrm{d}x}. \tag{37}$$

The introduction of a projection operator P for the functional derivative as suggested by Snyder et al.:[S1]

$$n_{i+1} = n_i - \eta P \left[\frac{\delta T[n]}{\delta n} + V - \mu \right], \tag{38}$$

yields a similar result for the Lagrange multiplier μ:

$$\mu = \frac{\int P\left[\frac{\delta T[n]}{\delta n} + V\right] \mathrm{d}x}{\int P[1] \, \mathrm{d}x}. \tag{39}$$

Note that the local principal component analysis (PCA) of Ref. S1 uses the difference of the density at the current step n_i and a subset of the training densities as basis for the projection operator. Since the integral over the difference of two valid densities is zero, the same will be true for all functions projected onto this subspace. The correct norm for the density n_{i+1} is therefore ensured without the need for a Lagrange multiplier.

5.2 Iteration on the Variable φ

In Section 3.3 and Section 3.4 of the main article a basis of sine functions is used to restrict the search space instead of the local PCA. This necessitates additionally enforcing a nonnegativity constraint, which is typically achieved by iterating on the variable $\varphi = \sqrt{n}$ instead of the density n. The steepest descent update rule for φ is given by:

$$\varphi_{i+1} = \varphi_i - \eta \frac{\delta \mathscr{L}}{\delta \varphi_i} = \varphi_i - \eta \frac{\delta \mathscr{L}}{\delta n_i} \frac{\delta n_i}{\delta \varphi_i} = \varphi_i - 2\eta \varphi_i \left(\frac{\delta T}{\delta n} + V - \mu \right), \tag{40}$$

where the Lagrange multiplier μ is again chosen such that the norm $\int n_{i+1} \, \mathrm{d}x = \int \varphi_{i+1}^2 \, \mathrm{d}x = N$ is ensured:

$$\begin{aligned} \int \varphi_{i+1}^2 \, \mathrm{d}x &= \int \left(\varphi_i - 2\eta \varphi_i \left(\frac{\delta T}{\delta n} + V - \mu \right) \right)^2 \mathrm{d}x \\ &= \int \varphi_i^2 \, \mathrm{d}x - 4\eta \int \varphi_i^2 \left(\frac{\delta T}{\delta n} + V - \mu \right) \mathrm{d}x + 4\eta^2 \int \varphi_i^2 \left(\frac{\delta T}{\delta n} + V - \mu \right)^2 \mathrm{d}x. \end{aligned} \tag{41}$$

Assuming that φ_i is properly normalized $\int \varphi_i^2 \, \mathrm{d}x = \int \varphi_{i+1}^2 \, \mathrm{d}x = N$ this simplifies to:

$$\begin{aligned} -4\eta \int \varphi_i^2 \left(\frac{\delta T}{\delta n} + V - \mu \right) \mathrm{d}x + 4\eta^2 \int \varphi_i^2 \left(\frac{\delta T}{\delta n} + V - \mu \right)^2 \mathrm{d}x &\overset{!}{=} 0 \\ \int \varphi_i^2 \left(\frac{\delta T}{\delta n} + V - \mu \right) \mathrm{d}x &= \eta \int \varphi_i^2 \left(\frac{\delta T}{\delta n} + V - \mu \right)^2 \mathrm{d}x. \end{aligned} \tag{42}$$

Since the step size η is an arbitrary positive number the integrals on both sides have to be equal to zero, which yields

$$\mu = \frac{\int \varphi_i^2 \left(\frac{\delta T}{\delta n} + V \right) \mathrm{d}x}{\int \varphi_i^2 \, \mathrm{d}x}. \tag{43}$$

Inspired by the local PCA method, the basis of sine functions is introduced using a projection operator acting on the functional derivative:

$$\varphi_{i+1} = \varphi_i - \eta P \left[\frac{\delta \mathscr{L}}{\delta \varphi_i} \right] = \varphi_i - \eta P \left[2\varphi_i \left(\frac{\delta T}{\delta n} + V - \mu \right) \right]. \tag{44}$$

The norm of n_{i+1} is then given by

$$\int \varphi_{i+1}^2 \, \mathrm{d}x = \int \varphi_i^2 \, \mathrm{d}x - 2 \int \eta \varphi_i P \left[2\varphi_i \left(\frac{\delta T}{\delta n} + V - \mu \right) \right] \mathrm{d}x + \int \eta^2 \left(P \left[2\varphi_i \left(\frac{\delta T}{\delta n} + V - \mu \right) \right] \right)^2 \mathrm{d}x. \tag{45}$$

Again assuming that $\int \varphi_i^2 \, \mathrm{d}x = \int \varphi_{i+1}^2 \, \mathrm{d}x = N$ yields:

$$\int \varphi_i P \left[\varphi_i \left(\frac{\delta T}{\delta n} + V - \mu \right) \right] \mathrm{d}x = \eta \int \left(P \left[\varphi_i \left(\frac{\delta T}{\delta n} + V - \mu \right) \right] \right)^2 \mathrm{d}x. \tag{46}$$

Using the same arguments as before and the fact that P is a linear operator the multiplier μ can be calculated via

$$\mu = \frac{\int \varphi_i P \left[\varphi_i \left(\frac{\delta T}{\delta n} + V \right) \right] \mathrm{d}x}{\int \varphi_i P [\varphi_i] \, \mathrm{d}x}. \tag{47}$$

6 Kernel Ridge Regression with Offset

The main problem of the iterative calculation of minimum energy densities is that the search algorithm is likely to leave the valid region, i.e. the region where the machine learning model offers correct predictions. Snyder et al.[S1] solved this by restricting the search space using a projection onto the subspace of training densities. Denzel and Kästner[S8] suggested a different solution for a similar problem in machine learning accelerated molecular geometry optimization. The extrapolation behavior of kernel based machine learning models can be tuned by introducing a constant offset:

$$T^{\mathrm{ML}}(\mathbf{n}) = b + \sum_{j}^{M} \alpha_j k(\mathbf{n}_j, \mathbf{n}). \tag{48}$$

In Figure S11 the effect of this offset term is demonstrated for a one-dimensional example. The models predict an output value of b for examples far away from the training data. Using larger values, therefore, offers the possibility of introducing an energy penalty for these out-of-training examples and thereby ensuring that the iterative search is restricted to the region spanned by the training examples. Note that this one-dimensional model trained on $M = 4$ examples does not display the problems encountered in the $G = 500$ dimensional model trained on just $M = 100$ examples discussed in the main article.

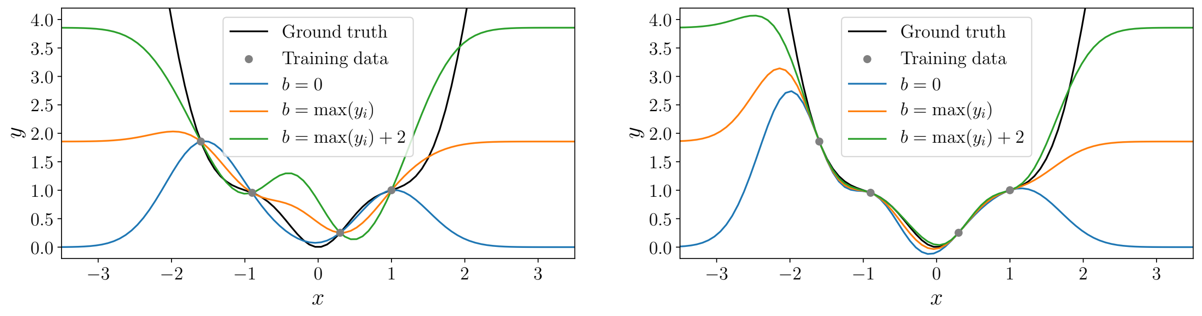

Figure S11: Kernel ridge regression of the function $y(x) = (x + \frac{1}{4}\sin(\pi x))^2$ using different offset values b. The model shown in the left panel is trained only on function values y_i, whereas the training data for the model in the right panel also includes derivative information $\mathrm{d}y_i/\mathrm{d}x$. Both models use a length scale of $\sigma = 0.5$ and parameters $\kappa = 1$ and $\lambda = 10^{-8}$.

As a first test the model errors on the $N = 1$ data set are evaluated. The results of a small grid search for the parameters $b \in \{\max(T_j), \max(T_j)+5, \max(T_j)+10\}$ and $\sigma \in \{2.5, 5.0, 10.0\}$ are summarized in Table S5. All presented models use the extended KRR formalism and $\kappa = 1$, $\lambda = 10^{-12}$ for the remaining hyperparameters. Table S5 shows that the largest value for the length scale parameter of $\sigma = 10.0$ yields the best accuracy for both the kinetic energy and the functional derivative. The value of the offset parameter b has a smaller influence on the model performance than the length scale parameter.

Table S5: Absolute error values on the test set for extended KRR models with offset term given in kcal/mol.

		$\|\Delta T\|$			$\|\Delta \frac{\delta T}{\delta n}\|$		
b	σ	mean	std	max	mean	std	max
$\max(T_j)$	2.5	0.256	2.253	61.617	55.932	95.658	1746.875
$\max(T_j)$ + 5	2.5	1.005	7.484	158.321	84.234	225.494	3929.710
$\max(T_j)$ + 10	2.5	2.260	16.813	338.897	183.468	477.609	8025.145
$\max(T_j)$	5.0	0.014	0.098	2.502	20.638	34.946	391.505
$\max(T_j)$ + 5	5.0	0.006	0.033	0.732	17.325	28.796	339.674
$\max(T_j)$ + 10	5.0	0.012	0.102	2.162	14.902	25.382	306.731
$\max(T_j)$	10.0	0.002	0.010	0.165	6.650	9.340	105.226
$\max(T_j)$ + 5	10.0	0.002	0.008	0.130	6.520	9.148	103.361
$\max(T_j)$ + 10	10.0	0.001	0.007	0.111	6.394	8.972	101.919

The results for the iteratively found densities are summarized in Table S6. All three models with length scale $\sigma = 10.0$ show a large number of minimization runs that leave the region spanned by the training examples, leading to extremely large mean absolute error values. While the results for $\sigma = 2.5$ suggest a better behavior during the minimization (at least for $b = \max(T_j)+5$ and $b = \max(T_j)+10$), the final errors achieved by these models are clearly limited by the poor model performance shown in Table S5. A length scale of $\sigma = 5.0$, therefore, represents a balanced trade-off between model error and the ability to restrict the search space. In fact, the performance for $b = \max(T_j)+10$ and $\sigma = 5.0$ is comparable to the results achieved using the principal component analysis presented in the main article. However, in general this method is not practical for large scale applications as it is difficult to determine a set of hyperparameters during training which will lead to models suitable for a usage in iterative minimizations. This problem does not arise in the original application of this approach in geometry optimization tasks as the training set is constantly extended by additional *ab-initio* calculations during minimization.

Table S6: Absolute kinetic energy error values for the iteratively found densities in kcal/mol as well as the integrated absolute error of the densities.

		$\|\Delta T\|$			$\|\Delta n\| \times 10^4$		
b	σ	mean	std	max	mean	std	max
$\max(T_j)$	2.5	15.143	52.469	381.315	17861.5	55933.6	259048.3
$\max(T_j) + 5$	2.5	2.884	12.282	209.400	13.5	38.8	534.0
$\max(T_j) + 10$	2.5	4.364	16.793	265.146	20.2	49.9	607.7
$\max(T_j)$	5.0	258.283	105.156	514.125	6199.3	7862.7	25692.3
$\max(T_j) + 5$	5.0	0.128	0.838	17.815	2.4	5.5	80.5
$\max(T_j) + 10$	5.0	0.075	0.642	15.196	1.2	3.5	71.0
$\max(T_j)$	10.0	5468.034	257.717	5958.720	5674.8	392.5	7273.1
$\max(T_j) + 5$	10.0	3787.131	243.600	4265.972	5226.8	405.1	6817.0
$\max(T_j) + 10$	10.0	2365.444	215.082	2814.292	4748.9	417.8	6292.2

7 Influence of the Local PCA on the Functional Derivative

Figure S12 shows the effect of the local PCA projection on the functional derivative prediction of the various models on the sample potential of Figure 2 of the main article. The corresponding

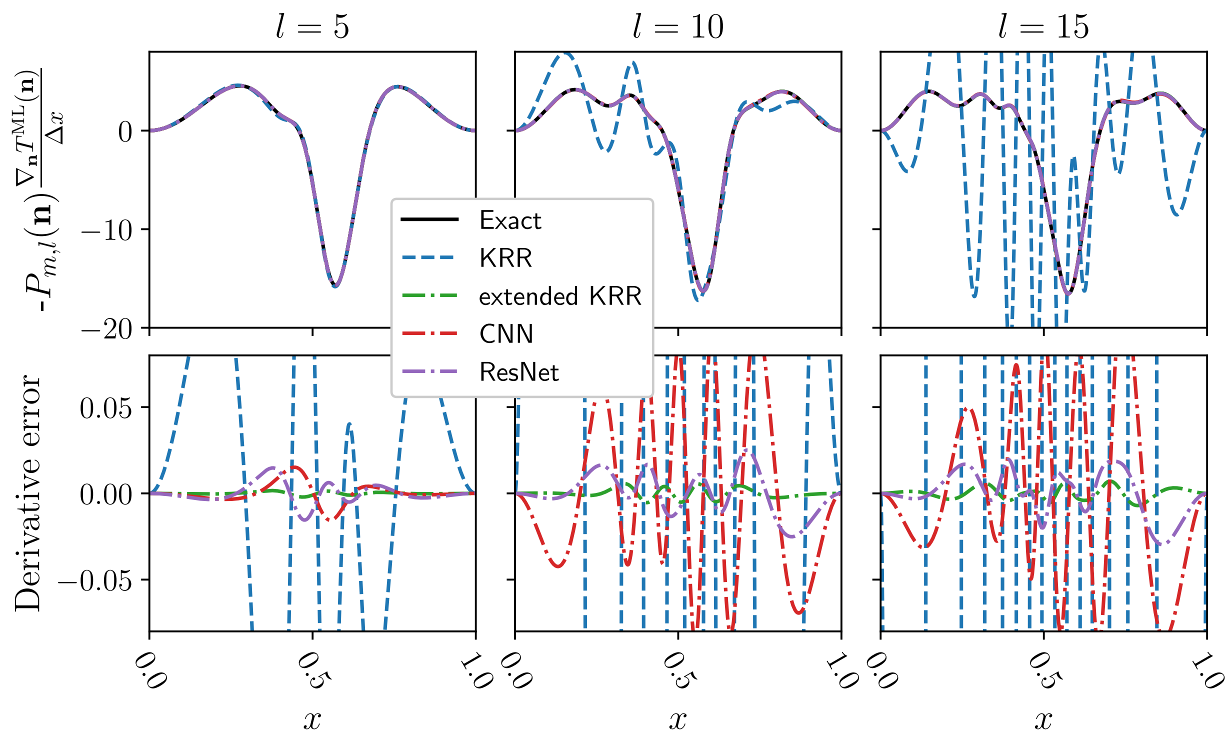

Figure S12: Projected functional derivative (top row) predicted by the machine learning models as well as the difference to the exact solution (bottom row) for the projection parameters $m = 30$ and various values of l (columns).

integrated absolute error values for the projected functional derivatives are summarized in Table S7. Note that both the exact and the machine learning predicted functional derivatives are projected.

Table S7: Absolute error values for the projected functional derivative on the $N = 1$ test set given in kcal/mol.

	$\lvert\Delta P_{m=30,l=5}(n)\frac{\delta T}{\delta n}\rvert$			$\lvert\Delta P_{m=30,l=10}(n)\frac{\delta T}{\delta n}\rvert$			$\lvert\Delta P_{m=30,l=15}(n)\frac{\delta T}{\delta n}\rvert$		
model	mean	std	max	mean	std	max	mean	std	max
KRR	147.5	177.8	1326.2	2008.1	879.6	4951.6	7874.8	3132.0	17518.6
ext. KRR	0.6	1.1	15.8	1.7	2.2	26.0	2.3	2.8	30.9
CNN	6.5	10.3	119.1	21.6	20.1	317.4	25.2	22.0	359.5
ResNet	1.9	2.5	54.1	4.7	4.4	81.4	7.6	5.8	85.0

8 Learning Curve

Figure S13 shows the learning curve, that is the accuracy of the models as a function of the number of the training examples, for the presented machine learning models. The errors are evaluated on the $N = 1$ test set while the first $M = 40$, 60, 80, and 100 training examples are used as training examples.

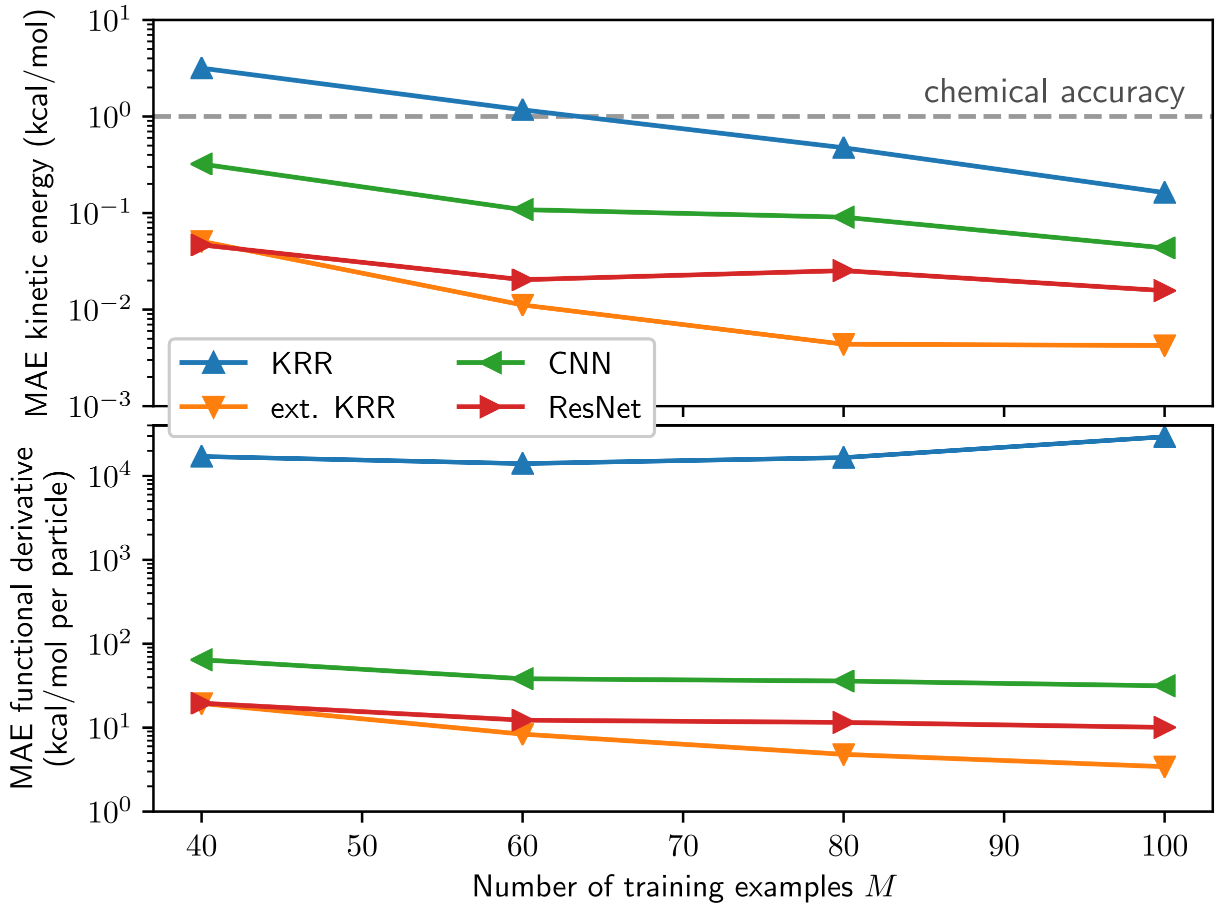

Figure S13: Errors on the $N = 1$ test set achieved by the various models as a function of the number of training examples M.

We use the hyperparameters given by Snyder et al. for the standard KRR method. A description of the method for optimizing the extended KRR hyperparameters as well as a detailed analysis of the results are given in Section 3. The hyperparameters for both KRR models are summarized in Table S8. For all of the neural network training runs the same hyperparameters as presented in

Section 4 is used and the batch size is reduced to the number of training examples M.

Figure S13 shows a general trend of decreasing error with increasing number of training examples. Once again the advantage of including derivative information becomes apparent as all three models trained on derivatives achieve chemical accuracy for the smallest training set size $M = 40$.

Note the slight increase in the kinetic energy error for the ResNet when the number of training examples M is increased from 60 to 80. This variation in the performance is to be expected due to the random initialization of the weights at the beginning of the training process. A detailed comparison of neural network learning curves could therefore only be done after averaging the results over several training runs.

Table S8: Hyperparameters used in the training of the KRR models for different training set sizes M.

	KRR		ext. KRR	
M	σ	$\lambda \times 10^{14}$	σ	$\lambda \times 10^{12}$
40	238	57 600	61.49	10
60	95	10 000	35.16	10
80	48	4489	30.58	1
100	43	12	30.58	1

9 Computational Timing

Figure S14 shows the computational time required by the various models for evaluating a single density of the test set. All times are measured by averaging over 100 runs on a workstation equipped with an Intel i7-920 and without GPU acceleration for tensorflow. Note that the individual evaluation times could be improved significantly by further optimization in the computational implementation and more specialized hardware. Nevertheless, this graph clearly shows that the computational effort for the kernel based methods increases with the number of training examples while the neural network models maintain a constant evaluation time.

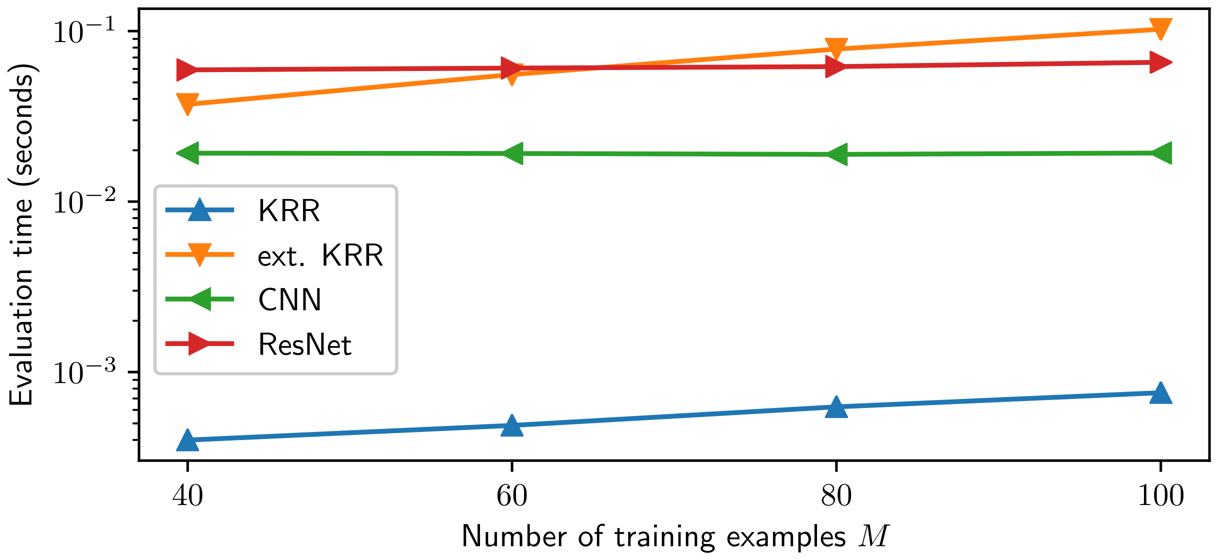

Figure S14: Evaluation time of the various models averaged over 100 runs as a function of the number of training examples.

References

(S1) Snyder, J. C.; Rupp, M.; Hansen, K.; Müller, K.-R.; Burke, K. Finding Density Functionals with Machine Learning. *Phys. Rev. Lett.* **2012**, *108*, 253002.

(S2) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.;

Brucher, M.; Perrot, M.; Duchesnay, E. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.

(S3) Duncan, W. LXXVIII. Some devices for the solution of large sets of simultaneous linear equations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **1944**, *35*, 660–670.

(S4) Guttman, L. Enlargement Methods for Computing the Inverse Matrix. *The Annals of Mathematical Statistics* **1946**, *17*, 336–343, Equation (14).

(S5) Henderson, H.; Searle, S. On Deriving the Inverse of a Sum of Matrices. *SIAM Rev.* **1981**, *23*, 53–60, Equation (13).

(S6) Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. 2014; `http://arxiv.org/abs/1412.6980`, Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

(S7) Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015; `http://tensorflow.org/`, Software available from tensorflow.org.

(S8) Denzel, A.; Kästner, J. Gaussian process regression for geometry optimization. *J. Chem. Phys.* **2018**, *148*, 094114.