Kohn-Sham equations as regularizer: building prior knowledge into machine-learned physics

Li Li (李力)[1],* Stephan Hoyer[1], Ryan Pederson[2], Ruoxi Sun (孙若溪)[1], Ekin D. Cubuk[1], Patrick Riley[1], and Kieron Burke[2,3]
[1] *Google Research, Mountain View, CA 94043, USA*
[2] *Department of Physics and Astronomy, University of California, Irvine, CA 92697, USA*
[3] *Department of Chemistry, University of California, Irvine, CA 92697, USA*

(Dated: September 21, 2020)

Including prior knowledge is important for effective machine learning models in physics, and is usually achieved by explicitly adding loss terms or constraints on model architectures. Prior knowledge embedded in the physics computation itself rarely draws attention. We show that solving the Kohn-Sham equations when training neural networks for the exchange-correlation functional provides an implicit regularization that greatly improves generalization. Two separations suffice for learning the entire one-dimensional H_2 dissociation curve within chemical accuracy, including the strongly correlated region. Our models also generalize to unseen types of molecules and overcome self-interaction error.

Differentiable programming [1] is a general paradigm of deep learning, where parameters in the computation flow are trained by gradient-based optimization. Based on the enormous development in automatic differentiation libraries [2–5], hardware accelerators [6] and deep learning [7], this emerging paradigm is relevant for scientific computing. It keeps rigorous components where we have extremely strong physics prior knowledge and well-established numerical methods [8] and parameterizes the approximation by a neural network, which can approximate any continuous function [9]. Recent highlights include discretizing partial differential equations [10], structural optimization [11], sampling equilibrium configurations [12], differentiable molecular dynamics [13], differentiable programming tensor networks [14], optimizing basis sets in Hartree-Fock [15] and variational quantum Monte Carlo [16–18].

Density functional theory (DFT), an approach to electronic structure problems, took an enormous step forward with the creation of the Kohn-Sham (KS) equations [19], which greatly improves accuracy from the original DFT [20–22]. The results of solving the KS equations are reported in tens of thousands of papers each year [23]. Given an approximation to the exchange-correlation (XC) energy, the KS equations are solved self-consistently. Results are limited by the quality of such approximations, and a standard problem of KS-DFT is to calculate accurate bond dissociation curves [24]. The difficulties are an example of strong correlation physics as electrons localize on separate nuclei [25].

Naturally, there has been considerable interest in using machine learning (ML) methods to improve DFT approximations. Initial work [26, 27] focused on the KS kinetic energy, as a sufficiently accurate approximation would allow by-passing the solving of the KS equations [28, 29]. For XC, recent works focus on learning the XC potential (not functional) from inverse KS [30], and use it in the KS-DFT scheme [31–34]. An important step forward was made last year, when it was shown that a neural network could find functionals using only three molecules, by training on both energies and densities [35], obtaining accuracy comparable to human-designed functionals, and generalizing to yield accurate atomization energies of 148 small molecules [36]. But this pioneering work does not yield chemical accuracy, nor approximations that work in the dissociation limit. Moreover, it uses gradient-free optimization which usually suffers from poor convergence behavior on the large number of parameters used in modern neural networks [37–39].

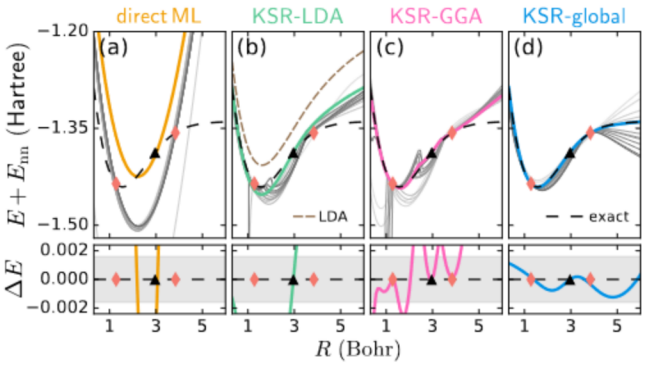

FIG. 1. One-dimensional H_2 dissociation curves trained from two molecules (red diamonds). (a) A ML model that directly predicts E from geometries, clearly fails to capture the physics from very limited data. (b) Comparison of LDA found with KSR and that from uniform gas (brown), and (c) same as (b) but for GGA, (d) the global XC approximation found with KSR. E_{nn} is the nucleus-nucleus repulsion energy. Grey lines denote 15 sampled functionals during training, with darker lines denoting later samples. Functionals with optimal parameters validated from the molecule at $R = 3$ (black triangles) are highlighted in orange, green, pink and blue respectively in each panel. KSR-global yields chemical accuracy (grey shadow), shown in lower panels. Atomic units used throughout. See the supplemental material [40] for more details.

Here, we show that all these limitations are overcome by incorporating the KS equations themselves into the neural network training by backpropagating through their iterations – a *KS regularizer* (KSR) to the ML model. In a traditional KS calculation, the XC is given, the equations are cycled to self-consistency, and all previous iterations are ignored in the final answer. In other ML work, functionals are trained on either energies alone [41–44], or even densities [32, 33, 45], but only after convergence. By incorporating the KS equations into the training, thereby learning the relation between density and energy at every iteration, we find accurate models with very little data and much greater generalizability.

Our results are illustrated in Figure 1, which is for a one-dimensional mimic of H_2 designed for testing electronic structure methods [46]. The distribution of curves of the ML model directly predicting E from geometries (direct ML) in (a) clearly fails to capture the physics. For local density approximation (LDA) and generalized gradient approximation (GGA) calculations similar to Nagai *et al.* [35] in (b-c), the effect of the KSR yields reasonably accurate results in the vicinity of the data, but not outside. But when a global XC functional is included in (d), chemical accuracy is achieved for all separations including the dissociation limit. Similar results can be achieved for H_4, the one-electron self-interaction error can easily be made to vanish, and the interaction of a pair of H_2 molecules can be found without any training on this type of molecule (all discussed below).

Modern DFT finds the ground-state electronic density by solving the Kohn-Sham equations:

$$\left\{ -\frac{\nabla^2}{2} + v_{\rm S}[n](\mathbf{r}) \right\} \phi_i(\mathbf{r}) = \epsilon_i \phi_i(\mathbf{r}). \tag{1}$$

The electronic density is obtained from occupied orbitals $n(\mathbf{r}) = \sum_i |\phi_i(\mathbf{r})|^2$. Here $v_{\rm S}[n](\mathbf{r}) = v(\mathbf{r}) + v_{\rm H}[n](\mathbf{r}) + v_{\rm XC}[n](\mathbf{r})$ is the KS potential consisting of the external one-body potential and the density-dependent Hartree (H) and XC potentials. The XC potential $v_{\rm XC}[n](\mathbf{r}) = \delta E_{\rm XC}/\delta n(\mathbf{r})$ is the functional derivative of the XC energy functional $E_{\rm XC}[n] = \int \epsilon_{\rm XC}[n](\mathbf{r}) n(\mathbf{r}) d\mathbf{r}$, where $\epsilon_{\rm XC}[n](\mathbf{r})$ is the XC energy per electron. The total electronic energy E is then given by the sum of the non-interacting kinetic energy $T_s[n]$, the external one-body potential energy $V[n]$, the Hartree energy $U[n]$, and XC energy $E_{\rm XC}[n]$.

The KS equations are in principle exact given the exact XC functional [19, 47], which in practice is the only term approximated in DFT. From a computational perspective, the eigenvalue problem of Eq. (1) is solved repeatedly until the density converges to a fixed point, starting from an initial guess. We use linear density mixing [48] to improve convergence, $n^{\rm (in)}_{k+1} = n^{\rm (in)}_k + \alpha(n^{\rm (out)}_k - n^{\rm (in)}_k)$. Figure 2(a) shows the unrolled computation flow. We approximate the XC energy per electron using a neural network $\epsilon_{{\rm XC},\theta}[n]$, where θ represents the trainable parameters. Together with the self-consistent KS iterations in Figure 2(b), the combined computational graph resembles a recurrent neural network [49] or deep equilibrium model [50] with additional fixed computational components. Density mixing has the same form as residual connections in deep neural networks [51]. In addition to improving convergence for the forward problem of KS self-consistent calculations, density mixing helps backpropagate gradients efficiently through long computational procedures.

If the neural XC functional were exact, KS self-consistent calculations would output the exact density and the intermediate energies over iterations would converge to the exact energy. This intention can be translated into a loss function and the neural XC functional can be updated end-to-end by backpropagating through the KS self-consistent calculations. This procedure differentiates through KS calculations and is general regardless of the dimensionality of the system. Throughout, experiments are performed in one dimension where accurate quantum solutions could be relatively easily generated via density matrix renormalization group (DMRG) [52]. The electron-electron repulsion is $A\exp(-\kappa|x-x'|)$, and attraction to a nucleus at $x=0$ is $-A\exp(-\kappa|x|)$ [40]. We design the loss function as an expectation $\mathbb{E}$ over

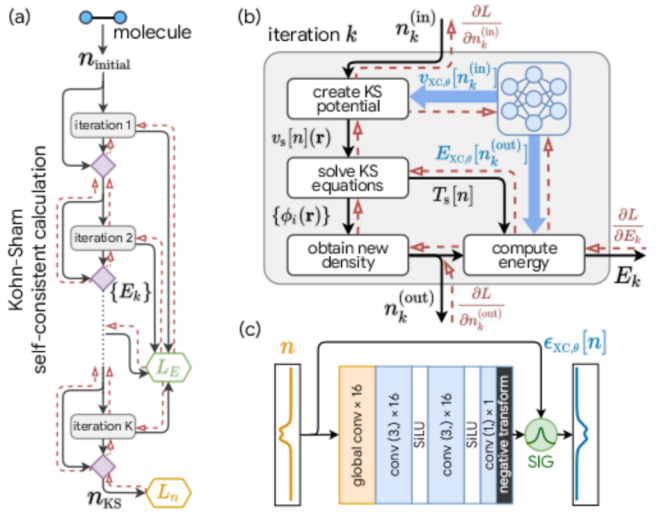

FIG. 2. (a) KS-DFT as a differentiable program. Black arrows are the conventional computation flow of KS self-consistent calculations with linear density mixing (purple diamonds). The gradients flow along red dashed arrows to minimize the energy loss L_E (green hexagon) and density loss L_n (orange hexagon). (b) In each single KS iteration, neural XC functional produces $v_{{\rm XC},\theta}[n]$ and $E_{{\rm XC},\theta}[n]$. (c) Architecture of global XC functional $\epsilon_{{\rm XC},\theta}[n]$.

training molecules,

$$L(\theta) = \underbrace{\mathbb{E}_{\text{train}}\left[\int dx (n_{\text{KS}} - n_{\text{DMRG}})^2 / N_e\right]}_{\text{density loss } L_n} + \underbrace{\mathbb{E}_{\text{train}}\left[\sum_{k=1}^{K} w_k (E_k - E_{\text{DMRG}})^2 / N_e\right]}_{\text{energy loss } L_E}, \quad (2)$$

where N_e is the number of electrons. L_n minimizes the difference between the final density with the exact density. L_E optimizes the trajectory of energies in total K iterations. The neural XC functional needs to not only output accurate ϵ_{XC} in each iteration, but also drive the iterations to quickly converge to the exact energy. The trajectory loss also makes backpropagation more efficient by directly flowing gradients to early iterations [53]. w_k are arbitrary non-negative weights associated with each iteration. The optimal neural network parameters are selected with minimal mean absolute energy per electron on the validation set.

Hundreds of useful XC functional approximations have been proposed by humans [54]. Here we build a neural XC functional with several differentiable components with physics intuition tailored for XC in Figure 2(c). A global convolution layer captures the long range interaction, $G(n(x), \xi_p) = \frac{1}{2\xi_p}\int dx' n(x') \exp(-|x - x'|/\xi_p)$. Note two special cases retrieve known physics quantities, Hartree energy density $G(n(x), \kappa^{-1}) \propto \epsilon_{\text{H}}$ and electronic density $G(n(x), 0) = n(x)$. Global convolution contains multiple channels and ξ_p of each channel is trainable to capture interaction in different scales. Although the rectified linear unit [55] is popular, we use the sigmoid linear unit (SiLU) [56] (also known as swish [57]) $f(x) = x/(1 + \exp(-x))$ because the infinite differentiability of SiLU guarantees the smoothness of v_{XC}, the first derivative, and the second and higher order derivatives of the neural network used in the L-BFGS training [58]. We do not enforce a specific choice of ϵ_{XC} (sometimes called a gauge [59]), but we do enforce some conditions, primarily to aid convergence of the algorithm. We require ϵ_{XC} to vanish whenever the density does, and that it be negative if at all possible. We achieved the former using the linearity of SiLU near the origin and turning off the bias terms in convolution layers. We softly impose the latter by a negative transform layer at the end, where a negative SiLU makes most output values negative. Finally, we design a self-interaction gate (SIG) that mixes in a portion of $-\epsilon_{\text{H}}$ to cancel the self-interaction error, $\epsilon_{\text{XC}}^{(\text{out})} = \epsilon_{\text{XC}}^{(\text{in})}(1 - \beta) - \epsilon_{\text{H}}\beta$. The portion is a gate function $\beta(N_e) = \exp(-(N_e - 1)^2/\sigma^2)$. When $N_e = 1$, then $\epsilon_{\text{XC}}^{(\text{out})} = -\epsilon_{\text{H}}$. For more electrons, σ can be fixed or adjusted by the training algorithm to decide the sensitivity to N_e. For H_2 as $R \to \infty$, ϵ_{XC} tends to a superposition of the negative of the Hartree energy density at each nucleus and approaches half that for H_2^+.

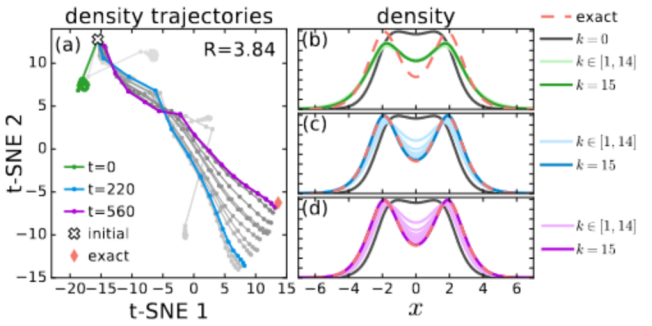

FIG. 3. (a) t-SNE visualization [60] of density trajectories (grey dots) sampled by KSR during training for $R = 3.84$ from initial guess (cross) to exact density (red diamond). Darker trajectories denote later optimization steps t. Densities from each KS step in trajectories are plotted in the corresponding highlighted colors for (b) untrained $t = 0$, (c) optimal $t = 220$ in Figure 1, and (d) overfitting $t = 560$.

Now we dive deeper into the outstanding generalization we observed in a simple but not easy task: predicting the entire H_2 dissociation curve from two points, as shown in Figure 1. It is not surprising that direct ML model completely fails. Neural networks are usually underdetermined systems as there are more parameters than training examples. Regularization is crucial to improve generalization [61, 62], especially when data is limited. Most existing works regularize models with particular physics prior knowledge by imposing *constraints* via feature engineering and preprocessing [63, 64], constraints on the network [65–68] or physics-informed loss terms [69, 70]. Another regularization strategy is to generate extra data for training using prior knowledge: in image classification problems, data are augmented by operations like flipping and cropping given the prior knowledge that labels are invariant to those operations [71]. However, it is not clear how to generate extra data for physics problems solved by specific methods, e.g. electronic structure problems with KS equations. We found that training from differentiating through KS self-consistent calculations regularizes the model. Although the exact densities and energies of only two separations are given, KSR naturally samples different trajectories from an initial density to the exact density at each training step. More importantly, KSR focuses on learning an XC functional that can lead the KS self-consistent calculations to converge to the exact density from the initial density. Figure 3 visualizes the density trajectories sampled by KSR for one training separation $R = 3.84$. The functional with untrained parameters ($t = 0$) samples densities near the initial guess but soon learns to explore broadly and finds the trajectories toward the vicinity of the exact density.

In contrast, most existing ML functionals learn to predict a single step from the exact density, which is a poor

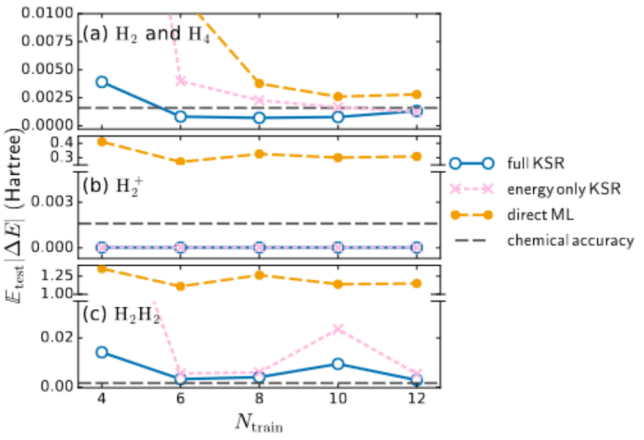

FIG. 4. Test generalization of models as a function of the total number of training examples N_{train}: full KSR (blue), energy only KSR (pink) and direct ML (orange) on (a) hold-out H_2 and H_4, and unseen types of molecules (b) H_2^+ (c) H_2H_2. Black dashed lines show chemical accuracy. See the supplemental material [40] for KS calculations and training details.

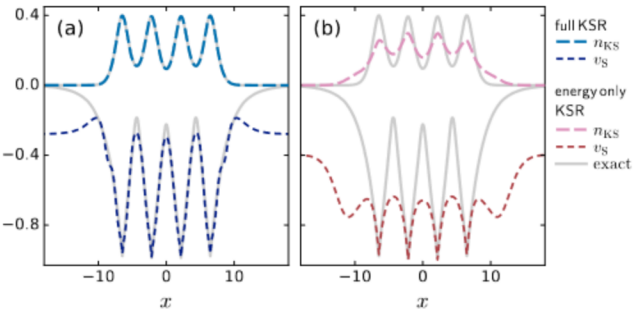

FIG. 5. Density and KS potential of H_4 with $R = 4.32$ from neural XC functionals trained with (a) full KSR and (b) energy only KSR on training set of size $N_{\text{train}} = 20$. v_{S} are shifted by a constant for better comparison.

surrogate for the full self-consistent calculations [72]. These standard ML models have two major shortcomings. First, the exact density is unknown for new systems, so the model is not expected to behave correctly on unseen initial densities for KS calculations. Second, even if a model is trained on many densities for single step prediction, it is not guaranteed to converge the self-consistent calculations to a good solution. Research in imitation learning shows that error accumulation from single steps quickly pushes the model out of its interpolation region [73]. On the other hand, since KSR allows the model access to all the KS iterations, it learns to optimize the entire self-consistent procedure to avoid the error accumulation from greedy optimization of single steps. Further comparison for training neural XC functionals without or with "weaker" KSR is in the supplemental material [40].

Next we retrain our neural XC functional with KSR on $N_{\text{train}}/2$ examples each of H_2 and H_4 molecules. Figure 4 shows the prediction accuracy of KSR with both energy and density loss (full KSR), in comparison to KSR with only energy loss (energy only KSR) and direct ML model. We compute the energy mean absolute error on the hold-out sets of H_2 ($R \in [0.4, 6]$) and H_4 ($R \in [1.04, 6]$). The average mean absolute error of H_2 and H_4 with various N_{train} is shown in Figure 4(a). Full KSR has the lowest error at minimum $N_{\text{train}} = 4$, reaching chemical accuracy at 6. As the size of the training set increases, energy only KSR reaches chemical accuracy at $N_{\text{train}} = 10$, but direct ML model never does (even at 20). Then we test models on unseen types of molecules. In Figure 4(b), both KSR models have perfect prediction on H_2^+ ($R \in [0.64, 8.48]$) because of the self-interaction gate in the neural XC functionals, while direct ML models always have large errors. Finally we take a pair of equilibrium H_2 and separate them with $R = 0.16$ to 9.76 Bohr, denoted as H_2H_2. KSR models generalize much better than ML for "zero-shot" prediction [74], where H_2H_2 have never been exposed to the model during training.

Why is the density important in training, and what use are the non-converged iterations? The density is the functional derivative of the energy with respect to the potential, so it gives the exact slope of the energy with respect to any change in the potential, including stretching (or compressing) the bond. Thus the density implicitly contains much energetic information, and reproducing the density accurately guarantees finding the correct derivative at that point in the binding curve. Iteration of the KS equations before convergence produces abundant information about the functional in the vicinity of the minimum, exploring its shape. Even before θ is close to its final value, the network is learning to construct a functional with both the correct minimum and all correct derivatives at this minimum. In the paradigm of differentiable programming, density is the hidden state carrying the information through the recurrent structure in Figure 2(a). Correct supervision from L_n greatly helps generalization from very limited data, see $N_{\text{train}} \leq 6$ in Figure 4. But as N_{train} increases, both KSR with and without L_n perform well in energy prediction. To address more details, we show the solution of H_4 with $R = 4.32$ in Figure 5. With L_n, the density is clearly much accurate than KSR without L_n ($\int (n_{\text{KS}} - n_{\text{DMRG}})^2 dx = 9.2 \times 10^{-5}$ versus 9.8×10^{-2}). Then we compute the corresponding exact v_{S} using inverse KS method [30]. Both functionals do not reproduce the exact v_{S}. However, functional trained with L_n recovered most of the KS potential. We must stress the nontriviality because unlike previous works [32–34] that explicitly included the KS or XC potential into the loss function, our model never uses the exact KS potential. In our KSR setup, the model aims at predicting ϵ_{XC}, from which the derived v_{S} yields accu-

rate density. Therefore, predicting $v_{\rm XC}$ is a side product. We also address some concerns on training explicitly with $v_{\rm XC}$. One artifact is that generating the exact $v_{\rm S}$ requires an additional inverse calculation, which is known to be numerically unstable [30]. Schmidt *et al.* [32] observe outliers when they generate training $v_{\rm XC}$ from inverse KS. While $v_{\rm XC}$ is a fascinating and useful object for theoretical study, because its relation to the density is extremely delicate, it is far more practical to simply use the density to train on [35].

Differentiable programming blurs the boundary between physics computation and ML. Here we showed that treating KS self-consistent calculations as a differentiable program is a regularizer, incorporating a physics prior and resulting in a remarkable generalization of the neural XC functional trained with it. The results serve as a proof of principle to rethink physics computation in the context of the new era of computing owing to achievements in automatic differentiation software, hardware and theories. An exciting next step is to apply this idea to real molecules, as an end-to-end differentiable electronic structure method. Besides finding density functionals, all heuristics in the calculations, e.g. initial guess, density update, preconditioning, basis sets, even the entire self-consistent calculations as a meta-optimization problem [53], can be learned and optimized while keeping the rigorous physics and mathematics in the rest of the algorithm – getting the best of both worlds.

The authors thank Michael Brenner, Sam Schoenholz, Lucas Wagner, and Hanjun Dai for helpful discussion. K.B. supported by NSF CHE-1856165, R.P. by DOE DE-SC0008696.

* email: leeley@google.com

[1] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, The Journal of Machine Learning Research **18**, 5595 (2017).

[2] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, "JAX: composable transformations of Python+NumPy programs," `http://github.com/google/jax` (2018).

[3] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah, CoRR (2018), arXiv:1811.01457.

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, in *Advances in neural information processing systems* (2019) pp. 8026–8037.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," (2015), software available from tensorflow.org.

[6] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, in *Proceedings of the 44th Annual International Symposium on Computer Architecture* (2017) pp. 1–12.

[7] Y. LeCun, Y. Bengio, and G. Hinton, nature **521**, 436 (2015).

[8] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt, (2019), arXiv:1907.07587 [cs.PL].

[9] K. Hornik, Neural networks **4**, 251 (1991).

[10] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, Proceedings of the National Academy of Sciences , 201814058 (2019).

[11] S. Hoyer, J. Sohl-Dickstein, and S. Greydanus, arXiv preprint arXiv:1909.04240 (2019).

[12] F. Noé, S. Olsson, J. Köhler, and H. Wu, Science **365**, eaaw1147 (2019).

[13] S. S. Schoenholz and E. D. Cubuk, "JAX M.D.: End-to-end differentiable, hardware accelerated, molecular dynamics in pure python," `https://github.com/google/jax-md`, `https://arxiv.org/abs/1912.04232` (2019), arXiv:1912.04232 [stat.ML].

[14] H.-J. Liao, J.-G. Liu, L. Wang, and T. Xiang, Physical Review X **9**, 031041 (2019).

[15] T. Tamayo-Mendoza, C. Kreisbeck, R. Lindh, and A. Aspuru-Guzik, ACS Cent Sci **4**, 559 (2018).

[16] J. Hermann, Z. Schätzle, and F. Noé, (2019), arXiv:1909.08423 [physics.comp-ph].

[17] D. Pfau, J. S. Spencer, A. G. de G. Matthews, and W. M. C. Foulkes, (2019), arXiv:1909.02487 [physics.chem-ph].

[18] L. Yang, Z. Leng, G. Yu, A. Patel, W.-J. Hu, and H. Pu, Phys. Rev. Research **2**, 012039 (2020).

[19] W. Kohn and L. J. Sham, Physical review **140**, A1133 (1965).

[20] P. Hohenberg and W. Kohn, Physical review **136**, B864 (1964).

[21] L. H. Thomas, in *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 23 (Cambridge University Press, 1927) pp. 542–548.

[22] E. Fermi, Rend. Accad. Naz. Lincei **6**, 5 (1927).

[23] R. O. Jones, Reviews of modern physics **87**, 897 (2015).

[24] E. M. Stoudenmire, L. O. Wagner, S. R. White, and K. Burke, Phys. Rev. Lett. **109**, 056402 (2012).

[25] A. J. Cohen, P. Mori-Sánchez, and W. Yang, Science **321**, 792 (2008).

[26] L. Li, J. C. Snyder, I. M. Pelaschier, J. Huang, U.-N. Niranjan, P. Duncan, M. Rupp, K.-R. Müller, and K. Burke, International Journal of Quantum Chemistry **116**, 819 (2016).

[27] J. C. Snyder, M. Rupp, K. Hansen, K.-R. Müller, and K. Burke, Physical review letters **108**, 253002 (2012).

[28] F. Brockherde, L. Vogt, L. Li, M. E. Tuckerman, K. Burke, and K.-R. Müller, Nature communications **8**, 1 (2017).

[29] L. Li, T. E. Baker, S. R. White, K. Burke, *et al.*, Physical Review B **94**, 245129 (2016).

[30] D. S. Jensen and A. Wasserman, International Journal of Quantum Chemistry **118**, e25425 (2018).

[31] D. J. Tozer, V. E. Ingamells, and N. C. Handy, The

Journal of Chemical Physics **105**, 9200 (1996).
[32] J. Schmidt, C. L. Benavides-Riveros, and M. A. Marques, The journal of physical chemistry letters **10**, 6425 (2019).
[33] Y. Zhou, J. Wu, S. Chen, and G. Chen, The journal of physical chemistry letters **10**, 7264 (2019).
[34] R. Nagai, R. Akashi, S. Sasaki, and S. Tsuneyuki, The Journal of chemical physics **148**, 241737 (2018).
[35] R. Nagai, R. Akashi, and O. Sugino, npj Computational Materials **6**, 1 (2020).
[36] L. A. Curtiss, K. Raghavachari, G. W. Trucks, and J. A. Pople, The Journal of chemical physics **94**, 7221 (1991).
[37] J. C. Duchi, M. I. Jordan, M. J. Wainwright, and A. Wibisono, IEEE Transactions on Information Theory **61**, 2788 (2015).
[38] N. Maheswaranathan, L. Metz, G. Tucker, D. Choi, and J. Sohl-Dickstein, in *International Conference on Machine Learning* (PMLR, 2019) pp. 4264–4273.
[39] L. M. Rios and N. V. Sahinidis, Journal of Global Optimization **56**, 1247 (2013).
[40] See Supplemental Material below for information about 1d model systems; computational details of DMRG; KS calculations; training, validation and test; neural networks; training without KSR and training with weaker KSR.
[41] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, in *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (JMLR. org, 2017) pp. 1263–1272.
[42] K. Schütt, P.-J. Kindermans, H. E. S. Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, in *Advances in neural information processing systems* (2017) pp. 991–1001.
[43] J. Behler and M. Parrinello, Physical review letters **98**, 146401 (2007).
[44] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, Physical review letters **108**, 058301 (2012).
[45] J. R. Moreno, G. Carleo, and A. Georges, Physical Review Letters **125**, 076402 (2020).
[46] T. E. Baker, E. M. Stoudenmire, L. O. Wagner, K. Burke, and S. R. White, Physical Review B **91**, 235141 (2015).
[47] L. O. Wagner, E. M. Stoudenmire, K. Burke, and S. R. White, Physical review letters **111**, 093003 (2013).
[48] G. Kresse and J. Furthmüller, Physical review B **54**, 11169 (1996).
[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, Tech. Rep. (California Univ San Diego La Jolla Inst for Cognitive Science, 1985).
[50] S. Bai, J. Z. Kolter, and V. Koltun, in *Advances in Neural Information Processing Systems (NeurIPS)* (2019).
[51] K. He, X. Zhang, S. Ren, and J. Sun, in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 770–778.
[52] S. R. White, Physical review letters **69**, 2863 (1992).
[53] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, in *Advances in neural information processing systems* (2016) pp. 3981–3989.
[54] N. Mardirossian and M. Head-Gordon, Molecular Physics **115**, 2315 (2017).
[55] V. Nair and G. E. Hinton, in *ICML* (2010).
[56] S. Elfwing, E. Uchibe, and K. Doya, Neural Networks **107**, 3 (2018).
[57] P. Ramachandran, B. Zoph, and Q. V. Le, arXiv preprint arXiv:1710.05941 (2017).
[58] D. C. Liu and J. Nocedal, Mathematical programming **45**, 503 (1989).
[59] J. P. Perdew, A. Ruzsinszky, J. Sun, and K. Burke, The Journal of chemical physics **140**, 18A533 (2014).
[60] L. v. d. Maaten and G. Hinton, Journal of machine learning research **9**, 2579 (2008).
[61] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) `http://www.deeplearningbook.org`.
[62] J. Kukačka, V. Golkov, and D. Cremers, arXiv preprint arXiv:1710.10686 (2017).
[63] E. D. Cubuk, A. D. Sendek, and E. J. Reed, The Journal of chemical physics **150**, 214701 (2019).
[64] J. Hollingsworth, L. Li, T. E. Baker, and K. Burke, The Journal of Chemical Physics **148**, 241743 (2018).
[65] N. Thomas, T. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley, arXiv preprint arXiv:1802.08219 (2018).
[66] K. Schütt, M. Gastegger, A. Tkatchenko, K.-R. Müller, and R. J. Maurer, Nature communications **10**, 1 (2019).
[67] R. Kondor, H. T. Son, H. Pan, B. Anderson, and S. Trivedi, arXiv preprint arXiv:1801.02144 (2018).
[68] S. Seo and Y. Liu, arXiv preprint arXiv:1902.02950 (2019).
[69] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Journal of Computational Physics **378**, 686 (2019).
[70] R. Sharma, A. B. Farimani, J. Gomes, P. Eastman, and V. Pande, arXiv preprint arXiv:1807.11374 (2018).
[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Advances in neural information processing systems* (2012) pp. 1097–1105.
[72] G. Tucker, A. Mnih, C. J. Maddison, J. Lawson, and J. Sohl-Dickstein, in *Advances in Neural Information Processing Systems* (2017) pp. 2627–2636.
[73] S. Ross and D. Bagnell, in *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010) pp. 661–668.
[74] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae, *et al.*, arXiv preprint arXiv:2004.10746 (2020).

Supplemental Material
Building prior knowledge into machine-learned physics: Kohn-Sham equations as a regularizer

Li Li (李力)[1],* Stephan Hoyer[1], Ryan Pederson[2], Ruoxi Sun (孙若溪)[1], Ekin D. Cubuk[1], Patrick Riley[1], and Kieron Burke[2,3]
[1] *Google Research, Mountain View, CA 94043, USA*
[2] *Department of Physics and Astronomy, University of California, Irvine, CA 92697, USA*
[3] *Department of Chemistry, University of California, Irvine, CA 92697, USA*

CONTENTS

* email: leeley@google.com

I. 1D MODEL SYSTEMS

In 1D, we utilize exponential Coulomb interactions to mimic the standard 3D Coulomb potential,

$$v_{\rm exp}(x) = A\exp(-\kappa|x|), \tag{S1}$$

where $A = 1.071295$ and $\kappa^{-1} = 2.385345$ [1]. Within this model, the external one-body potential for a nuclei of atomic number Z and position x' is represented by $-Z\,v_{\rm exp}(x-x')$. The external potential for arbitrary molecular systems and geometries is modeled as

$$v(x) = -\sum_j Z_j v_{\rm exp}(x - x_j)\,. \tag{S2}$$

For example, a 1D H_2 molecule at separation $R = 4$ can be represented by $v(x) = -v_{\rm exp}(x-2) - v_{\rm exp}(x+2)$. The repulsion between electrons at positions x and x' is given by the two-body potential $v_{\rm ee}(x-x') = v_{\rm exp}(x-x')$. We represent all systems on a 1D grid of $m = 513$ points each separated by a distance $h = 0.08$. The center grid point is at the origin, $x = 0$, and the range of grid points is $x \in \{-20.48, \ldots, 20.48\}$. For consistency, all nuclei positions reside on grid points in calculations. In this convention, all molecules in this work are either symmetric about the origin $x = 0$ or $x = 0.04$, depending on the separation between nuclei.

II. DMRG CALCULATION DETAILS

The real-space interacting Hamiltonian for a 1D system of lattice spacing h becomes in second quantized notation,

$$H = \frac{5}{4h^2}\sum_{j,\sigma} n_{j\sigma} - \frac{2}{3h^2}\sum_{\langle i,j\rangle,\sigma} c^\dagger_{i\sigma}c_{j\sigma} + \frac{1}{24h^2}\sum_{\langle\langle i,j\rangle\rangle,\sigma} c^\dagger_{i\sigma}c_{j\sigma} + \sum_j v(x_j)\,n_j + \sum_{ij} v_{\rm ee}(x_i - x_j)\,n_i\,n_j\,, \tag{S3}$$

where the operator $c^\dagger_{j\sigma}$ creates (and $c_{j\sigma}$ annihilates) an electron of spin σ on site j, $n_{j\sigma} = c^\dagger_{j\sigma}c_{j\sigma}$, and $n_j = n_{j\uparrow} + n_{j\downarrow}$. The single and double brackets below the sums indicate sums over nearest and next nearest neighbors, respectively. The hopping term coefficients are determined by the 4-th order central finite difference approximation to the second derivative. The Hamiltonian is solved using DMRG to obtain highly accurate ground-state energies and densities. Calculations are performed using the ITensor library [2] with an energy convergence threshold of 10^{-7} Ha.

III. KS CALCULATION DETAILS

A. Local Density Approximation

In our 1D model the electron repulsion is an exponential interaction. To implement a local density approximation (LDA) for this interaction we use Ref. [1] which provides the exponentially repelling uniform gas exchange energy analytically and an accurate parameterized model for the correlation energy. We use this specific implementation for all LDA calculations.

B. Initial density

We solve the Schrödinger equation of the non-interacting system with the external potential $v(x)$ defined in Eq. S2,

$$\left\{-\frac{\nabla^2}{2} + v(x)\right\}\phi_i(x) = \epsilon_i\,\phi_i(x). \tag{S4}$$

The density is the square sum of all the occupied orbitals $n(x) = \sum_i |\phi_i(x)|^2$. In all the KS self-consistent calculations presented in this work, we use the density of the non-interacting system with external potential $v(x)$ as the initial density.

C. Linear density mixing

Linear density mixing is a well-known strategy to improve the convergence of the KS self-consistent calculation,

$$n_{k+1}^{(\mathrm{in})} = n_k^{(\mathrm{in})} + \alpha(n_k^{(\mathrm{out})} - n_k^{(\mathrm{in})}). \tag{S5}$$

In this work, we apply an exponential decay on the mixing factor $\alpha = 0.5 \times 0.9^{k-1}$.

D. Symmetry

The training molecules used in this paper are symmetric to their centers. We define the symmetry operation on functions on the grids $\mathcal{S} : \mathbb{R}^m \to \mathbb{R}^m$. It flips a function at the center and averages with itself. In each KS iteration, we enforce the symmetry on the XC functional, $\epsilon_{\mathrm{XC}}[n] \to \mathcal{S}\big(\epsilon_{\mathrm{XC}}[\mathcal{S}(n)]\big)$. So E_{XC} and v_{XC} are transformed as

$$E_{\mathrm{XC}} = \int n\,\epsilon_{\mathrm{XC}}[n]dx \to \int n\,\mathcal{S}\big(\epsilon_{\mathrm{XC}}[\mathcal{S}(n)]\big)dx \tag{S6}$$

$$v_{\mathrm{XC}} = \frac{\partial \int n\,\epsilon_{\mathrm{XC}}[n]dx}{\partial n} \to \frac{\partial \int n\,\mathcal{S}\big(\epsilon_{\mathrm{XC}}[\mathcal{S}(n)]\big)dx}{\partial n}. \tag{S7}$$

Before the output of each KS iteration, $n_k^{(\mathrm{out})} \to \mathcal{S}(n_k^{(\mathrm{out})})$.

IV. TRAINING, VALIDATION AND TEST

A. Weights in trajectory loss

We use $w_k = 0.9^{K-k}H(k-10)$, where H is the Heaviside function and K is the total number of iterations.

B. Number of KS iterations

All the KS calculations use a fixed number of iterations that is sufficient for convergence. The number of iterations for different molecules are listed in Table I.

TABLE I. Number of iterations for different molecules.

	H_2	H_4	H_2^+	H_2H_2
train	15	40	–	–
validation	15	40	–	–
test	15	40	5	10

C. Dataset for learning H_2 dissociation from two molecules

H_2 dissociation curves in Figure 1 are trained from exact densities and energies of two molecules. One is a compressed H_2 ($R = 1.28$) and the other is a stretched H_2 ($R = 3.84$) molecule. The optimal checkpoint is selected by a validation molecule with $R = 2.96$.

D. Dataset for learning and predicting several types of molecules

1. *Training molecules*

The distances between nearby atoms for training molecules used in Figure 4 are listed in Table II.

TABLE II. Distances between nearby atoms for training molecules used in Figure 4.

N_{train}	H_2										H_4									
4			1.28					3.84					2.08			3.36				
6	0.48		1.28					3.84			1.28		2.08			3.36				
8	0.48		1.28			3.04		3.84			1.28		2.08			3.36			4.48	
10	0.48		1.28			3.04		3.84		4.64	1.28		2.08			3.36		4.00	4.48	
12	0.48		1.28		2.40	3.04		3.84		4.64	1.28		2.08		3.04	3.36		4.00	4.48	
14	0.48		1.28		2.40	3.04	3.52	3.84		4.64	1.28		2.08		3.04	3.36	3.68	4.00	4.48	
16	0.48		1.28	1.76	2.40	3.04	3.52	3.84		4.64	1.28		2.08	2.56	3.04	3.36	3.68	4.00	4.48	
18	0.48		1.28	1.76	2.40	3.04	3.52	3.84	4.16	4.64	1.28		2.08	2.56	3.04	3.36	3.68	4.00	4.48	4.80
20	0.48	0.80	1.28	1.76	2.40	3.04	3.52	3.84	4.16	4.64	1.28	1.76	2.08	2.56	3.04	3.36	3.68	4.00	4.48	4.80

2. *Validation molecules*

The distances between nearby atoms for validation molecules used in Figure 4 are listed in Table III.

TABLE III. The distances between nearby atoms for validation molecules used in Figure 4. The validation set is fixed for calculations with $4 \leq N_{\text{train}} \leq 20$.

Molecule	R
H_2	1.68, 2.96, 4.40, 5.52
H_4	1.84, 2.64, 3.28, 5.04

3. *Test molecules*

The distances between nearby atoms for test molecules used in Figure 4 are listed in Table IV.

TABLE IV. The distances between nearby atoms for test molecules used in Figure 4. The test set is fixed for calculations with $4 \leq N_{\text{train}} \leq 20$.

Molecule	R
H_2	0.40, 0.56, 0.72, 0.88, 1.04, 1.20, 1.36, 1.52, 1.84, 2.00, 2.16, 2.32, 2.48, 2.64, 2.80, 3.12, 3.28, 3.44, 3.60, 3.76, 3.92, 4.08, 4.24, 4.56, 4.72, 4.88, 5.04, 5.20, 5.36, 5.68, 5.84, 6.00
H_4	1.04, 1.20, 1.36, 1.52, 1.68, 2.00, 2.16, 2.32, 2.48, 2.80, 2.96, 3.12, 3.44, 3.60, 3.76, 3.92, 4.08, 4.24, 4.40, 4.56 4.72, 4.88, 5.20, 5.36, 5.52, 5.68, 5.84, 6.00
H_2^+	0.64, 0.80, 0.96, 1.12, 1.28, 1.44, 1.60, 1.76, 1.92, 2.08, 2.24, 2.40, 2.48, 2.56, 2.64, 2.72, 2.88, 3.04, 3.20, 3.36, 3.52, 3.68, 3.84, 4.00, 4.16, 4.32, 4.48, 4.64, 4.80, 4.96, 5.12, 5.28, 5.44, 5.60, 5.76, 5.92, 6.08, 6.24, 6.40, 6.56, 6.72, 6.88, 7.04, 7.20, 7.36, 7.52, 7.68, 7.84, 8.00, 8.16, 8.32, 8.48
H_2H_2	0.16, 0.48, 0.80, 1.12, 1.44, 1.76, 2.08, 2.40, 2.72, 3.04, 3.36, 3.68, 4.00, 4.32, 4.64, 4.96, 5.28, 5.60, 5.92, 6.24, 6.56, 6.88, 7.20, 7.52, 7.84, 8.16, 8.48, 8.80, 9.12, 9.44, 9.76

4. *Test errors*

We extend the plot of test errors in Figure 4 to $N_{\text{train}} = 20$ in Figure S1 and list all the numerical values in Table V.

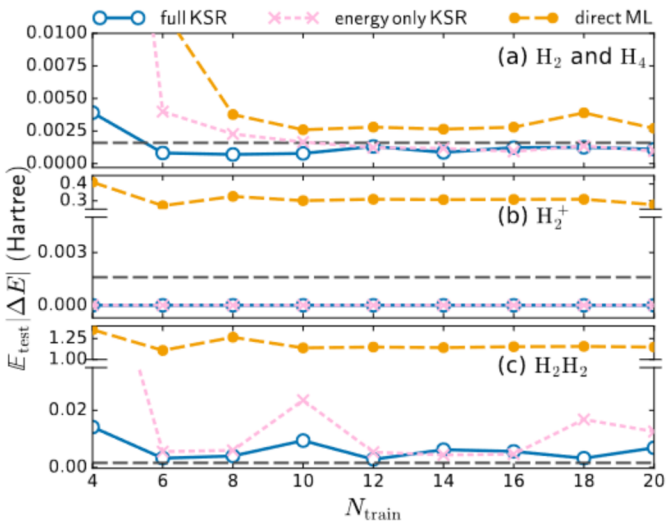

FIG. S1. Test generalization of ML models as a function of the total number of training examples N_{train}: full KSR (blue), energy only KSR (pink) and direct ML (orange) on (a) holdout H_2 and H_4, and unseen types of molecules (b) H_2^+ (c) H_2H_2. Black dashed lines show chemical accuracy. All the numerical values are listed in Table V.

TABLE V. Numerical values of test errors plotted in Figure S1.

Model	Molecule	N_{train} 4	6	8	10	12	14	16	18	20
KSR $L_n + L_E$	H_2 and H_4	3.91×10^{-3}	8.15×10^{-4}	7.04×10^{-4}	7.82×10^{-4}	1.32×10^{-3}	8.64×10^{-4}	1.23×10^{-3}	1.25×10^{-3}	1.11×10^{-3}
	H_2^+	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}
	H_2H_2	1.42×10^{-2}	3.23×10^{-3}	4.02×10^{-3}	9.41×10^{-3}	2.83×10^{-3}	6.23×10^{-3}	5.62×10^{-3}	3.23×10^{-3}	6.87×10^{-3}
KSR L_E	H_2 and H_4	4.82×10^{-2}	3.99×10^{-3}	2.27×10^{-3}	1.66×10^{-3}	1.23×10^{-3}	1.17×10^{-3}	9.37×10^{-4}	1.35×10^{-3}	9.73×10^{-4}
	H_2^+	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}	1.71×10^{-5}
	H_2H_2	9.19×10^{-2}	5.56×10^{-3}	5.99×10^{-3}	2.36×10^{-2}	5.35×10^{-3}	4.38×10^{-3}	4.67×10^{-3}	1.68×10^{-2}	1.26×10^{-2}
ML	H_2 and H_4	4.95×10^{-2}	1.18×10^{-2}	3.77×10^{-3}	2.60×10^{-3}	2.80×10^{-3}	2.65×10^{-3}	2.79×10^{-3}	3.89×10^{-3}	2.70×10^{-3}
	H_2^+	4.10×10^{-1}	2.71×10^{-1}	3.26×10^{-1}	3.01×10^{-1}	3.09×10^{-1}	3.07×10^{-1}	3.08×10^{-1}	3.09×10^{-1}	2.76×10^{-1}
	H_2H_2	1.36	1.11	1.27	1.14	1.15	1.14	1.15	1.16	1.15

V. NEURAL NETWORKS

A. Architecture

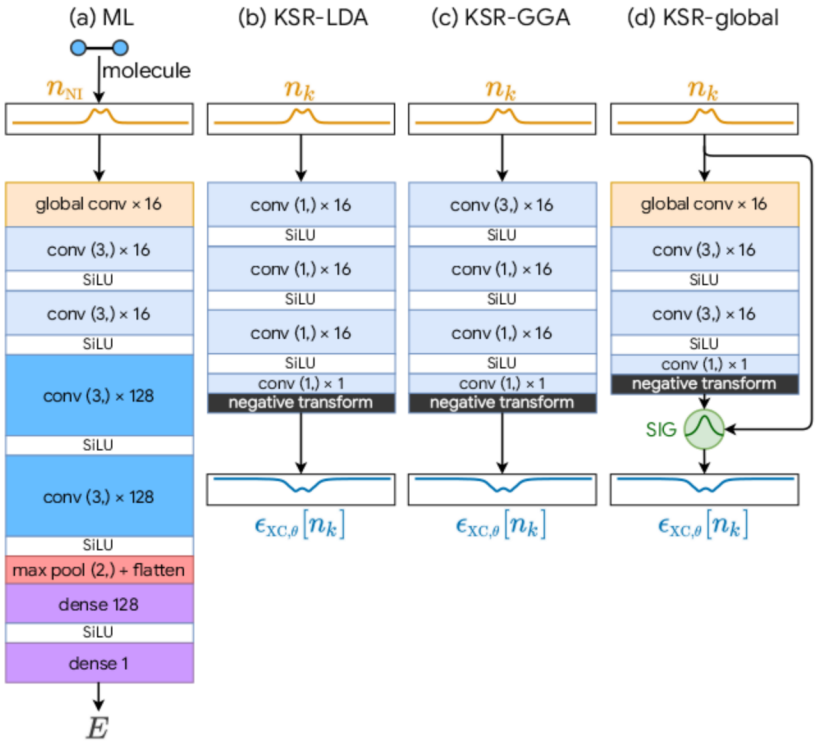

FIG. S2. Model architectures of (a) the ML model that directly predicts energy E from geometry, (b) the neural LDA with KSR, (c) the neural GGA with KSR, and (d) the neural global functional.

Figure S2 illustrates the model architectures used in Figure 1. In the ML model that directly predicts energy from geometry (Figure S2(a)), we first solve Eq. S4 to obtain a density for a particular molecular geometry. We use this density as a smooth representation of the geometry. The first few layers (global conv-conv-SiLU-conv-SiLU) are identical to the KSR-global in Figure S2(d). Next, we use convolution layers with 128 channels and dense layers to increase the capacity of the model. Finally, a dense layer with a single unit outputs the scalar E.

The KSR-LDA and KSR-GGA approaches do not have the global convolution layer because the use of global information violates the local and semi-local approximation. The first layer of KSR-LDA is a convolution with filter size 1. It mimics the physics of the standard LDA approach by mapping the density value to the XC energy density at the same point, $\epsilon^{\mathrm{LDA}}_{\mathrm{XC},\theta} : \mathbb{R}^1 \to \mathbb{R}^1$. KSR-GGA uses a convolution layer with filter size 3 to map the density values of three nearby points to the XC energy density at the center point, $\epsilon^{\mathrm{GGA}}_{\mathrm{XC},\theta} : \mathbb{R}^3 \to \mathbb{R}^1$. The XC energy density in the entire space is also computed pointwise, $\epsilon_{\mathrm{XC}} = \left\{\epsilon^{\mathrm{GGA}}_{\mathrm{XC},\theta}[n(x_{-1}, x_0, x_1)], \ldots, \epsilon^{\mathrm{GGA}}_{\mathrm{XC},\theta}[n(x_{m-2}, x_{m-1}, x_m)]\right\} \in \mathbb{R}^m$. The remaining network structure of KSR-LDA and KSR-GGA is identical to KSR-global, except for self-interaction gate (SIG).

B. Layers

1. *Convolution*

Filter weights in 1D convolution are initialized by He normal initialization [3]. The stride is one. The edges are padded with zero to ensure that the size of the output spatial dimension is the same as the size of the unpadded input spatial dimension. There is no bias term.

2. Global convolution

Global convolution contains multiple channels to capture the interaction in different scales. The operation in each channel is

$$G(n(x),\xi_p) = \frac{1}{2\xi_p}\int dx' n(x') \exp(-|x-x'|/\xi_p). \tag{S8}$$

where ξ_p is trainable and controls the scale of the interaction. We parameterize $\xi_p = a+(b-a)\cdot\sigma(\eta_p)$ using the sigmoid function $\sigma(x) = 1/(1+\exp(-x))$ to bound $\xi_p \in (a,b)$. η_p is initialized using the normal distribution $\mathcal{N}(0,10^{-4})$. For the 16-channel global convolution layer used in this work, we have $\eta_1 \equiv 0$ to preserve the input density and the rest $\eta_p \in (0.1,\kappa^{-1})$ are trainable.

3. Dense

The dense layer is only used in the ML model (Figure S2(a)). The weights are initialized using Glorot normal initialization [4] and bias terms are initialized using the normal distribution $\mathcal{N}(0,10^{-4})$.

C. Checkpoint selection

Each calculation is repeated with 25 random seeds. The model is trained by L-BFGS [5] implemented in SciPy [6] `scipy.optimize.fmin_l_bfgs_b` with `factr=1,m=20,pgtol=1e-14`. Parameter checkpoints are saved every 10 steps until L-BFGS stops. The optimal checkpoint is the checkpoint with the lowest average energy error per electron on validation sets, $\mathbb{E}_{\{\mathcal{S}_{\rm val}\}}\mathbb{E}_{\mathcal{M}\in\mathcal{S}_{\rm val}}|(E-E_{\rm DMRG})|/N_e$, where E is the final energy from KS calculations, $E_{\rm DMRG}$ is the exact energy, and N_e is the number of electrons. The validation sets $\{\mathcal{S}_{\rm val}\}$ and the molecules $\mathcal{M}$ in each sets are listed in Table III.

VI. TRAINING A NEURAL XC FUNCTIONAL WITHOUT KS REGULARIZATION

Schmidt *et al.* [7] proposed a neural XC functional that can be used in a KS self-consistent calculation in the inference stage. Unlike our work, which trains the network through KS calculations, they train the network in a single-step. The training set contains 12800 molecules and the validation set contains 6400 molecules. Here, the molecules are exact solutions of one-dimensional two-electron problems in the external potential of up to three random nuclei (Equation 4 in Schmidt *et al.* [7]). The exact $v_{\rm XC}$ for each molecule is computed by an inverse KS method. They input exact ground state density and train the network to predict XC energy per length $e_{\rm XC}$ that minimizes the loss function: a weighted combination of the mean square errors (MSE) of the XC energy, XC potential, its numerical spatial derivative, and the difference between the XC energy and the integral over the potential (Equation 5 in Schmidt *et al.* [7]),

$$L(\theta) = \alpha\,\mathrm{MSE}(E_{\rm XC}) + \beta\,\mathrm{MSE}(v_{\rm XC}) + \gamma\,\mathrm{MSE}\Big(\frac{dv_{\rm XC}(x)}{dx}\Big) + \delta\,\mathrm{MSE}\Big(E_{\rm XC} - \int dx\, v_{\rm XC}(x)n(x)\Big), \tag{S9}$$

where $\alpha = 1.0$, $\beta = 100.0$, $\gamma = 10.0$, and $\delta = 1.0$ is the weights used in Schmidt *et al.* [7].

It is natural to pose the question: does the generalization from the two H_2 training molecules in Figure 1 result from using KS self-consistent calculations in the inference stage rather than the training stage? This is a reasonable concern because the XC energy is usually a small portion of the total energy. To justify this concern, we first use inverse KS to get the exact $v_{\rm XC}$ on the two H_2 molecules used in the H_2 experiment. Then, we take the model architecture [8] and loss function in Schmidt *et al.* [7] and attempt to learn the entire dissociation curve of H_2 from two molecules. Figure S3 compares the results from KS self-consistent calculations using functionals trained on (a) single-step and (b) Kohn-Sham regularization. It is not surprising that even though both approaches use KS self-consistent calculations in the inference stage, the model trained on a single-step fails to generalize in the small training set limit (1/6400 training set size to the original paper). The neural XC functional is a many-to-many mapping, which is very hard to learn with limited data. Moreover, KS self-consistent calculations start with an initial density that is not the exact ground state density. It is clearly out of the interpolation region for the model that has only seen exact densities of two molecules.

We would like to emphasize that this comparison aims to show that using KS calculations in *training* – Kohn-Sham regularizer – is crucial to the generalization. A single-step model could work well as reported in Schmidt *et al.* [7] with a larger training set and exact $v_{\rm XC}$.

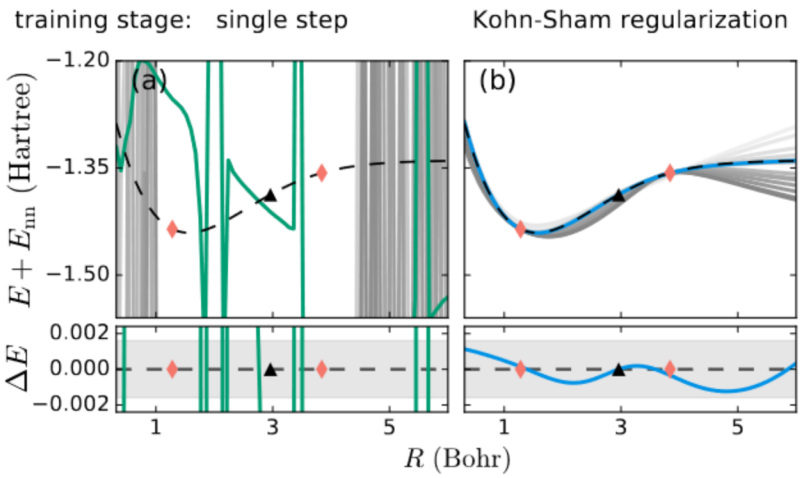

FIG. S3. Training the neural XC functional using (a) single-step and (b) Kohn-Sham regularization. Both functionals use Kohn-Sham self-consistent calculations in the inference stage.

VII. TRAINING A NEURAL XC FUNCTIONAL WITH "WEAKER" KOHN-SHAM REGULARIZATION

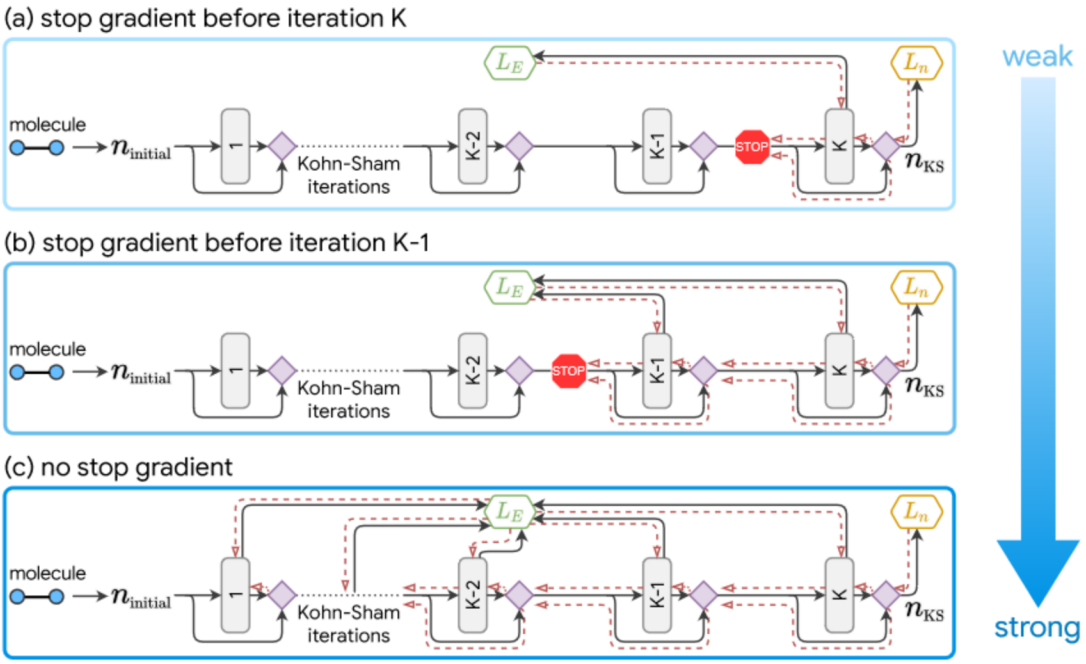

FIG. S4. Computational graph with different KSR strength. (a) stop gradient before iteration K. (b) stop gradient before iteration $K-1$. (c) no stop gradient. This is the same computation graph used in the main text.

Unlike other methods that build physics prior knowledge to the model through constraint, KSR "augments" densities for the model during training. Thus, there is no single coefficient to explicitly control the strength of the regularization. A straightforward idea to control the KSR strength is to change the total number of iterations K in the KS self-consistent calculations. However, a small K may not be sufficient to converge KS calculations. Thus it is ambiguous to understand whether the worse performance is from weaker regularization or unconverged KS calculations. Here we design an approach to control the strength of KSR by stopping the gradient flow in the backpropagation and keeping K fixed.

Stop gradient is a common operation in differentiable programming. It acts as an identity in the forward pass so it

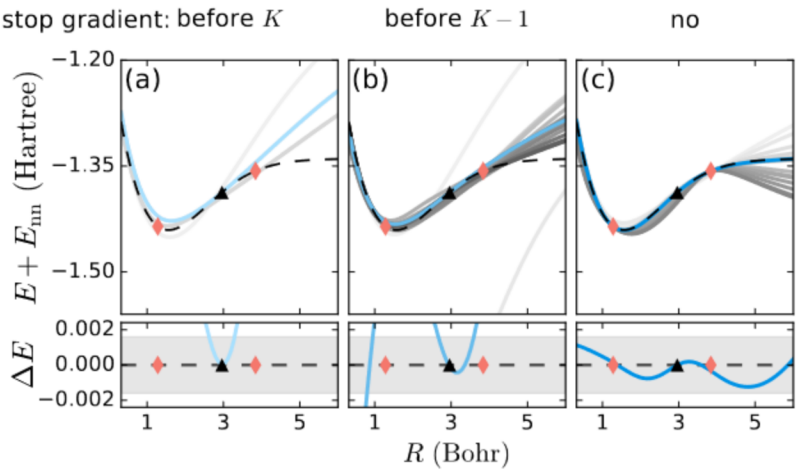

FIG. S5. H_2 dissociation curves trained from two molecules (red diamonds) with a corresponding computational graph shown in Figure S4.

does not affect the KS calculations. In the backpropagation, it sets the gradient passing through it to zero. As shown in Figure S4, we stop gradient before a certain KS iteration $k = k^*$ so all the previous iterations $k < k^*$ have no access to the gradient information. Since the gradient may still flow into the iteration k from L_E through its energy output E_k, we also stop the gradient on E_k for $k < k^*$. To simplify the graph, we remove the arrows between E_k to L_E for $k < k^*$. In Figure S4(a), the neural XC functional is updated only from the gradient information flowing through the final iteration. (b) is similar to (a) but has access to the gradient flowing through the last two iterations. No stop gradient is applied to (c) and it is identical to the computational graph we used in the main text. We repeat the same experiment in Figure 1. Figure S5 shows the H_2 dissociation curves trained with three stop gradient setting in Figure S4. In Figure S5(a), L-BFGS converges quickly as there is no sufficient gradient information for training. By including the gradient information in the $K-1$-th iteration, the distribution of the dissociation curves predicted by the model during training get closer to the true curve in (b). For comparison, we place the distribution of dissociation curves from model without stop gradient in (c), previously shown in Figure 1(d), where the physics of the true dissociation curve is captured.

[1] T. E. Baker, E. M. Stoudenmire, L. O. Wagner, K. Burke, and S. R. White, Physical Review B **91**, 235141 (2015).

[2] M. Fishman, S. R. White, and E. M. Stoudenmire, "The ITensor software library for tensor network calculations," (2020), arXiv:2007.14822.

[3] K. He, X. Zhang, S. Ren, and J. Sun, in *Proceedings of the IEEE international conference on computer vision* (2015) pp. 1026–1034.

[4] X. Glorot and Y. Bengio, in *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010) pp. 249–256.

[5] D. C. Liu and J. Nocedal, Mathematical programming **45**, 503 (1989).

[6] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, Nature Methods **17**, 261 (2020).

[7] J. Schmidt, C. L. Benavides-Riveros, and M. A. Marques, The journal of physical chemistry letters **10**, 6425 (2019).

[8] The only difference is that this model predicts $\epsilon_{\rm XC}[n](x)$ rather than $e_{\rm XC}[n](x)$ in Schmidt *et al.* [7]. The relation between them is $e_{\rm XC}[n](x) = \epsilon_{\rm XC}[n](x) \cdot n(x)$.