Learning the exchange-correlation functional from nature with differentiable density functional theory

Muhammad F. Kasim[1], **& Sam M. Vinko**[1,2]
[1]Department of Physics, University of Oxford, Oxford, OX1 3PU, UK
[2]Central Laser Facility, STFC Rutherford Appleton Laboratory, Didcot OX11 0QX, UK
{firstname.lastname}@physics.ox.ac.uk

Abstract

Improving the predictive capability of molecular properties in *ab initio* simulations is essential for advanced material discovery. Despite recent progress making use of machine learning, utilizing deep neural networks to improve quantum chemistry modelling remains severely limited by the scarcity and heterogeneity of appropriate experimental data. Here we show how training a neural network to replace the exchange-correlation functional within a fully-differentiable three-dimensional Kohn-Sham density functional theory (DFT) simulation can greatly improve its accuracy and generalizability. Using only eight experimental data points on diatomic molecules, our trained exchange-correlation networks provided improved predictions of atomization energies across a collection of 104 molecules containing new bonds and atoms that are not present in the training.

1 Introduction

Fast and accurate predictive models of atomic and molecular properties are keys in advancing novel material discovery. With the rapid development of machine learning (ML) techniques, considerable effort has been dedicated to accelerating simulations, enabling the efficient exploration of configuration space. Much of this work focuses on end-to-end ML approaches [1], where the models are trained to predict some specific molecular or material properties using tens to hundreds of thousands of data points, generated via simulations [2]. Once trained, ML models have the advantage of accurately predicting the simulation outputs much faster than the original simulations [3]. The level of accuracy achieved by ML models with respect to the simulated data can be very high, and may even reach chemical accuracy on select systems outside the original training dataset [4]. These machine learning approaches show great promise, but rely heavily on the availability of large datasets for training.

While there has been a concerted effort in accelerating simulations, less attention has been dedicated to using machine learning to increase the accuracy of the simulation itself with respect to the experimental data. In particular, as ML models typically rely on some underlying simulation for the training data, the accuracy of the trained models can never exceed the simulations in terms of accuracy. However, training ML models using experimental data is challenging as the available data is highly heterogeneous as well as limited, making it difficult to train accurate ML models tailored to reproduce a specific system or physical property. This limitation also poses a great challenge on the generalization capability of ML models to systems outside the training data.

One emerging solution to this problems is to use machine learning to build models which learn the exchange-correlation (xc) functional within the framework of Density Functional Theory (DFT) simulations [5, 6, 7, 8, 9, 10]. Here, the Kohn-Sham DFT (KS-DFT) [11] can be used to calculate various ground-state properties of molecules and materials, where its accuracy largely depends on the quality of the employed xc functional. Works along these lines in the one-dimensional case

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

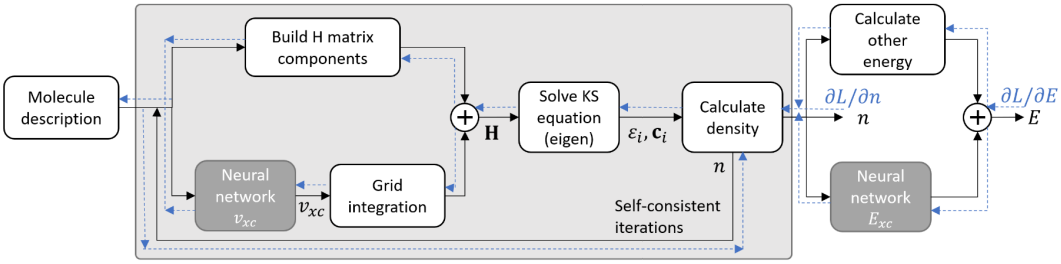

Figure 1: Schematics of the xc neural network and differentiable KS-DFT code. The trainable neural network is illustrated with dark gray boxes. The solid dark lines show the forward calculation flow while the dashed blue lines show the gradient backpropagation flow.

show promise [9, 10], but are severely limited by the lack of availability of experimental data from one-dimensional systems.

In this context, we present here a new machine learning approach to learn the xc functional using heterogeneous and limited experimental data. We achieve this in a hybrid of physics and machine learning way by incorporating a deep neural network representing the xc functional in a fully differentiable three-dimensional KS-DFT calculation. Notably, using a xc neural network (XCNN) in a fully differentiable KS-DFT calculation gives us a framework to train the xc functional using any available experimental properties, however heterogeneous, which can be simulated via a standard KS-DFT. Moreover, as the xc neural network is independent of the system (atom type, structure, number of electrons, *etc*.), a well-trained XCNN should be generalizable to a wide range of systems, including those outside the pool of training data. In this paper, we demonstrate this approach using a dataset comprising atoms and molecules from the first few rows of the periodic table, but the results are readily extendable to more complex structures.

2 Differentiable DFT

A single biggest challenge to realizing our method is writing a KS-DFT simulation in a fully differentiable manner. This is achieved here by writing the KS-DFT code using an automatic differentiation library, PyTorch [12], and by providing separately the gradients of all the KS-DFT components which are not available in PyTorch. The KS-DFT calculation involves self-consistent iterations of solving the Kohn-Sham equations,

$$H[n](\mathbf{r})\phi_i(\mathbf{r}) = \varepsilon_i\phi_i(\mathbf{r}), \tag{1}$$

$$n(\mathbf{r}) = \sum_i f_i|\phi_i(\mathbf{r})|^2, \tag{2}$$

where $n(\mathbf{r})$ is the electron density as a function of a 3D coordinate ($\mathbf{r}$), f_i is the occupation number of the i-th orbital (ϕ_i), ε_i is the Kohn-Sham orbital energy, and $H[n](\mathbf{r})$ is the Hamiltonian operator, a functional of electron density profile n and a function of the 3D coordinate. The Hamiltonian operator consists of kinetic, external potential, Hartree potential, and xc potential operators.

The KS-DFT calculation proceeds by solving Eq. (1) with eigendecompositions and computing Eq. (2) iteratively until convergence or self-consistency is reached. In this work, the Hamiltonian operator $H[n]$ is constructed using the contracted Gaussian-type basis [13] as the orbital basis. The schematics of our differentiable KS-DFT code is shown in Fig. 1. Unless specified otherwise, all calculations involved in this paper use 6-311++G(3df,3pd) basis sets [14]. More details on solving the KS equations can be found in the appendix.

Although modern automatic differentiation libraries provide many operations that enable an automatic propagation of gradient calculations, there are many gradient operations that have to be implemented specifically for this project. An important example is the self-consistent cycle iteration. Previous works on differentiable quantum chemistry [15, 10] employed a fixed number of linear mixing iterations to achieve self-consistency, and let the automatic differentiation library propagate the gradient through the chain of iterations. In contrast, we implement the gradient propagation of the

self-consistency cycle using an implicit gradient calculation (see appendix) [16, 17], allowing us to deploy a better self-consistent algorithm, achieving faster and more robust convergence than linear mixing.

Another example is the gradient calculation of the eigendecomposition with (near-)degenerate eigenvalues which often produces numerical instabilities in the gradient calculation. To solve this problem, we follow the calculation by Kasim [18] to obtain a numerically stable gradient calculation in the (near-)degenerate cases. Besides those two examples, there are more KS-DFT parts that require specific gradient implementations. The details of those parts can be found in the appendix.

With the fully differentiable KS-DFT code, the xc energy represented by a deep neural network can be trained efficiently. For this work we will test two different types of xc deep neural networks (XCNN) based on two rungs of Jacob's ladder – the Local Density Approximation (LDA) and the Generalized Gradient Approximation (GGA). These xc energies can be written as:

$$E_{\mathrm{nnLDA}}[n] = \alpha E_{\mathrm{LDA(xc)}}[n] + \beta \int n(\mathbf{r}) f(n, \xi)\, \mathrm{d}\mathbf{r} \tag{3}$$

$$E_{\mathrm{nnPBE}}[n] = \alpha E_{\mathrm{PBE(xc)}}[n] + \beta \int n(\mathbf{r}) f(n, \xi, s)\, \mathrm{d}\mathbf{r}, \tag{4}$$

where $E_{\mathrm{LDA(xc)}}$ is the LDA exchange [11] and correlation energy [19], and $E_{\mathrm{PBE(xc)}}$ is the PBE exchange-correlation energy [20]. Here α and β are two trainable parameters starting from 1 and 0 respectively, and $f(\cdot)$ is the trainable neural network. The LDA and PBE energies are calculated using Libxc [21], wrapped with PyTorch to provide the required gradient calculations. The XCNNs take inputs of the local electron density n, the relative spin polarization $\xi = (n_\uparrow - n_\downarrow)/n$, and the normalized density gradient $s = |\nabla n|^2/(24\pi^2 n^4)^{1/3}$.

For the XCNN training, we compiled a small dataset consisting of experimental atomization energies (AE) of diatomic molecules from the NIST CCCBDB database [22] and calculated density profiles using a CCSD calculation [23] as a regularizer [24]. The dataset is split into 2 groups: training and validation. The training dataset is used directly to update the parameters of the neural network via the gradient of a loss function (see appendix for details on the loss function). After the training is finished, we selected a model checkpoint during the training that gives the lowest loss function for the validation dataset.

The complete list of atoms and molecules used in the training and validation datasets is shown in Table 1. We note that some atoms are only present in the validation set and not in the training set, to obtain checkpoints that are able to generalize well to new types of atoms outside the training set.

3 Results

To test the performance of the trained XCNNs, we prepared a test dataset consisting of the atomization energies of 104 molecules from ref [25] and the ionization potentials of atoms H-Ar from NIST ASD [27]. The experimental geometric data as well as the atomization energies for the 104 molecules were obtained from the NIST CCCBDB database. There are actually 146 molecules in ref [25], however, only 104 molecules have experimental data in CCCBDB, so we limit our dataset to those. The atomization energy test dataset contains molecules with 2-11 atoms ranging from H-Cl, containing many bonds and atoms that do not present in the training and validation datasets. The molecules in the test dataset are listed in the appendix.

The results of training the XCNNs using the dataset are presented in Table 2. We see that in this case both XCNN-LDA and XCNN-PBE provide significant improvement over their bases, i.e. LDA and PBE, respectively. In terms of average atomization energy prediction errors across the 104 test molecules, XCNN-LDA achieves more than 4 times lower error than LDA (PW92) and XCNN-PBE

Table 1: Atoms and molecules presented in the training and validation datasets.

Type	Atomization energy	Density profile
Training	H_2, LiH, O_2, CO	H, Li, Ne, H_2, Li_2, LiH, B_2, O_2, CO
Validation	N_2, NO, F_2, HF	He, Be, N, N_2, F_2, HF

Table 2: Mean absolute error (MAE) in kcal/mol of the atomization energy and ionization potential for atoms and molecules in the test dataset. The column "IP" represents the deviation in ionization potential for atoms H-Ar. Column "AE" is the MAE of atomization energy of a collection of 104 molecules from ref [25]. The following 4 columns present the atomization energies of subsets of molecules from the full 104-molecule set: "AE HC" for hydrocarbons, "AE SHC" for substituted hydrocarbons, "AE NHC-1" for non-hydrocarbon molecules containing only first and second row atoms, and "AE NHC-2" for non-hydrocarbon molecules containing at least one third row atom. The bolded values are the best MAE in the respective column for each group of approximations.

Calculation	IP	AE	AE HC	AE SHC	AE NHC-1	AE NHC-2
Local Density Approximations (LDA)						
LDA [19]	**6.9**	70.4	97.0	101.1	58.8	43.7
XCNN-LDA	50.8	**15.3**	**22.7**	**19.4**	**7.8**	**16.6**
XCNN-LDA-IP	15.2	18.5	25.4	21.8	8.4	22.9
Generalized gradient approximations (GGA)						
PBE [20]	**3.6**	16.5	15.1	23.2	18.0	9.8
XCNN-PBE	10.7	**7.4**	**5.4**	**8.4**	**6.5**	**8.5**
XCNN-PBE-IP	4.1	8.1	6.8	9.6	6.7	8.6
Other approximations						
SCAN [26]	3.7	5.2	3.8	8.2	4.0	4.6
CCSD (basis: cc-pvqz)	2.0	12.1	11.1	17.7	11.2	9.0
CCSD-T (basis: cc-pvqz)	**1.3**	**3.5**	**2.2**	**5.6**	**2.5**	**3.5**

achieves more than 2 times lower error than PBE. Importantly, although the training and validation sets contain atomization energies only for 8 diatomic molecules, both XCNNs provide improvements on atomization energy predictions of the 104-molecule test dataset, which includes molecules with up to 11 atoms.

By looking into the subsets of the atomization energy dataset, both XCNN-LDA and XCNN-PBE also achieve better atomization energy predictions than their bases on all of the subsets. The substantial improvements seen on the hydrocarbon and substituted hydrocarbon molecules show the generalization capability of the technique to molecules with bonds not present in neither the training nor the validation dataset (e.g., C–H, C–C, C=C). Moreover, both XCNNs can also predict better atomization energy on the "AE NHC-2" subset containing third-row atoms (i.e. Na-Ar) that are not present in the training and validation datasets, showing great generalization capability of the method.

Despite these encouraging results on the atomization energies, both XCNNs lead to significantly worse predictions of atomic ionization potentials compared with their bases. Therefore, for the next experiment we trained the XCNN by adding ionization potential data into the training and validation datasets, as shown in Tab. 1. The xc neural networks trained with this additional data are called "XCNN-LDA-IP" and "XCNN-PBE-IP". Adding ionization potential data to the training and validation datasets improves the performance of XCNNs on the ionization potential at the cost of slightly increasing the error on the atomization energy predictions. Although the ionization potential predictions of XCNN-IPs do not have lower error compared to their bases, they produce considerably better predictions on the atomization energies of the tested molecules than their bases.

Although the trained XCNNs cannot make better predictions than higher level of approximations, e.g. SCAN [26] and CCSD(T), the presented method shows that significant improvement can be made within the same level of approximation. The level of accuracy can be improved in the future work by using a higher level of approximation, such as moving to the higher rungs of Jacob's ladder or incorporating more global density information with neural networks in calculating the xc energy.

4 Conclusion

We presented a novel approach to train machine-learned xc functionals within the framework of fully-differential Kohn-Sham DFT using experimental data. Our xc neural networks, trained on a few atoms and diatomic molecules, are able to improve the prediction of atomization and ionization energies across a set of 104 molecules not present in the training dataset, including larger molecules containing new types of bonds and atoms.

Broader impact

Computational quantum chemistry has been an active research field since decades ago with large potential in chemistry and material sciences. This paper presents a new way to solve the accuracy problem in computational quantum chemistry that potentially have a significant impact both positively and negatively. More accurate quantum chemistry calculation can accelerate materials discovery and whether it brings positive or negative impact, depends on the use of the materials. For example, if the computational quantum chemistry is used to find drugs for curing rare diseases, it would bring positive impact to the society. However, if it is used to find an explosive materials for weapons, it would be perceived negatively by most people.

Acknowledgments and Disclosure of Funding

We would like to thank Kyle Bystrom for a useful suggestion on compiling experimental data. M.F.K. and S.M.V. acknowledge support from the UK EPSRC grant EP/P015794/1 and the Royal Society. S.M.V. is a Royal Society University Research Fellow. The authors declare no conflict of interest.

References

[1] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

[2] Kristof T Schütt, Huziel E Sauceda, P-J Kindermans, Alexandre Tkatchenko, and K-R Müller. Schnet–a deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24):241722, 2018.

[3] Justin S Smith, Olexandr Isayev, and Adrian E Roitberg. Ani-1: an extensible neural network potential with dft accuracy at force field computational cost. *Chemical science*, 8(4):3192–3203, 2017.

[4] Felix A Faber, Luke Hutchison, Bing Huang, Justin Gilmer, Samuel S Schoenholz, George E Dahl, Oriol Vinyals, Steven Kearnes, Patrick F Riley, and O Anatole Von Lilienfeld. Prediction errors of molecular machine learning models lower than hybrid dft error. *Journal of chemical theory and computation*, 13(11):5255–5264, 2017.

[5] Ryo Nagai, Ryosuke Akashi, and Osamu Sugino. Completing density functional theory by machine learning hidden messages from molecules. *npj Computational Materials*, 6(1):1–8, 2020.

[6] John C Snyder, Matthias Rupp, Katja Hansen, Leo Blooston, Klaus-Robert Müller, and Kieron Burke. Orbital-free bond breaking via machine learning. *The Journal of chemical physics*, 139(22):224104, 2013.

[7] Jonathan Schmidt, Carlos L Benavides-Riveros, and Miguel AL Marques. Machine learning the physical nonlocal exchange–correlation functional of density-functional theory. *The journal of physical chemistry letters*, 10(20):6425–6431, 2019.

[8] Rodrigo A Vargas-Hernández. Bayesian optimization for calibrating and selecting hybrid-density functional models. *The Journal of Physical Chemistry A*, 124(20):4053–4061, 2020.

[9] John C Snyder, Matthias Rupp, Katja Hansen, Klaus-Robert Müller, and Kieron Burke. Finding density functionals with machine learning. *Physical review letters*, 108(25):253002, 2012.

[10] Li Li, Stephan Hoyer, Ryan Pederson, Ruoxi Sun, Ekin D Cubuk, Patrick Riley, Kieron Burke, et al. Kohn-sham equations as regularizer: Building prior knowledge into machine-learned physics. *Physical Review Letters*, 126(3):036401, 2021.

[11] Walter Kohn and Lu Jeu Sham. Self-consistent equations including exchange and correlation effects. *Physical review*, 140(4A):A1133, 1965.

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037. Curran Associates, Inc., 2019.

[13] Benjamin P Pritchard, Doaa Altarawy, Brett Didier, Tara D Gibson, and Theresa L Windus. New basis set exchange: An open, up-to-date resource for the molecular sciences community. *Journal of chemical information and modeling*, 59(11):4814–4820, 2019.

[14] Timothy Clark, Jayaraman Chandrasekhar, Günther W Spitznagel, and Paul Von Ragué Schleyer. Efficient diffuse function-augmented basis sets for anion calculations. iii. the 3-21+ g basis set for first-row elements, li–f. *Journal of Computational Chemistry*, 4(3):294–301, 1983.

[15] Teresa Tamayo-Mendoza, Christoph Kreisbeck, Roland Lindh, and Alán Aspuru-Guzik. Automatic differentiation in quantum chemistry with applications to fully variational hartree–fock. *ACS central science*, 4(5):559–566, 2018.

[16] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

[17] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021.

[18] Muhammad Firmansyah Kasim. Derivatives of partial eigendecomposition of a real symmetric matrix for degenerate cases. *arXiv preprint arXiv:2011.04366*, 2020.

[19] John P Perdew and Yue Wang. Accurate and simple analytic representation of the electron-gas correlation energy. *Physical review B*, 45(23):13244, 1992.

[20] John P Perdew, Kieron Burke, and Matthias Ernzerhof. Generalized gradient approximation made simple. *Physical review letters*, 77(18):3865, 1996.

[21] Miguel AL Marques, Micael JT Oliveira, and Tobias Burnus. Libxc: A library of exchange and correlation functionals for density functional theory. *Computer physics communications*, 183(10):2272–2281, 2012.

[22] Russel D Johnson III and NIST CCCBDB Team. NIST Computational Chemistry Comparison and Benchmark Database (Release 21), [Online]. Available: `https://dx.doi.org/10.18434/T47C7Z[2020, August]`. National Institute of Standards and Technology, Gaithersburg, MD., 2020.

[23] Jiří Čížek. On the correlation problem in atomic and molecular systems. calculation of wavefunction components in ursell-type expansion using quantum-field theoretical methods. *The Journal of Chemical Physics*, 45(11):4256–4266, 1966.

[24] Michael G Medvedev, Ivan S Bushmarinov, Jianwei Sun, John P Perdew, and Konstantin A Lyssenko. Density functional theory is straying from the path toward the exact functional. *Science*, 355(6320):49–52, 2017.

[25] Larry A Curtiss, Krishnan Raghavachari, Paul C Redfern, and John A Pople. Assessment of gaussian-2 and density functional theories for the computation of enthalpies of formation. *The Journal of Chemical Physics*, 106(3):1063–1079, 1997.

[26] Jianwei Sun, Adrienn Ruzsinszky, and John P Perdew. Strongly constrained and appropriately normed semilocal density functional. *Physical review letters*, 115(3):036402, 2015.

[27] A. Kramida, Yu. Ralchenko, J. Reader, and and NIST ASD Team. NIST Atomic Spectra Database (ver. 5.8), [Online]. Available: `https://dx.doi.org/10.18434/T4W30F[2020, October]`. National Institute of Standards and Technology, Gaithersburg, MD., 2020.

[28] Clemens Carel Johannes Roothaan. New developments in molecular orbital theory. *Reviews of modern physics*, 23(2):69, 1951.

[29] Qiming Sun. Libcint: An efficient general integral library for g aussian basis functions. *Journal of computational chemistry*, 36(22):1664–1671, 2015.

[30] Saswata Dasgupta and John M Herbert. Standard grids for high-precision integration of modern density functionals: Sg-2 and sg-3. *Journal of computational chemistry*, 38(12):869–882, 2017.

[31] Axel D Becke. A multicenter numerical integration scheme for polyatomic molecules. *The Journal of chemical physics*, 88(4):2547–2553, 1988.

[32] Henk A Van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing*, 13(2):631–644, 1992.

[33] Qiming Sun, Timothy C Berkelbach, Nick S Blunt, George H Booth, Sheng Guo, Zhendong Li, Junzi Liu, James D McClain, Elvira R Sayfutyarova, Sandeep Sharma, et al. Pyscf: the python-based simulations of chemistry framework. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 8(1):e1340, 2018.

[34] Richard M Martin. *Electronic structure: basic theory and practical methods*. Cambridge university press, 2020.

[35] Karl K Irikura. Experimental vibrational zero-point energies: Diatomic molecules. *Journal of physical and chemical reference data*, 36(2):389–397, 2007.

[36] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[37] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See the last paragraph in section 3.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See "Broader impact" section.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Appendix A.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] For data splits, see Table 1 and Appendix E. For hyperparameters, see see Appendix D3.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D3.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See Appendix A.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Code availability

The differentiable quantum chemistry (DQC) code used for this paper can be found in https://github.com/diffqc/dqc/ with Apache-2.0 license. The DQC code uses PyTorch as well as xitorch, an additional differentiable scientific library written specifically for this project. The repository for xitorch can be found in https://github.com/xitorch/xitorch/ with MIT license.

B KS-DFT Calculation

The Hamiltonian operator in Eq. (1) is given in atomic units as $H[n](\mathbf{r}) = -\nabla^2/2 + v(\mathbf{r}) + v_H[n](\mathbf{r}) + v_{xc}[n](\mathbf{r})$ with $-\nabla^2/2$ representing the kinetic operator, $v(\mathbf{r})$ the external potential operator including ionic Coulomb potentials, $v_H[n](\mathbf{r})$ the Hartree potential operator, and $v_{xc}[n](\mathbf{r})$ the xc potential operator. The xc potential operator is defined as the functional derivative of the xc energy with respect to the density, i.e. $v_{xc}[n](\mathbf{r}) = \delta E_{xc}[n]/\delta n(\mathbf{r})$, which can be calculated using an automatic differentiation library.

To discretize the Hamiltonian operator, a contracted Gaussian basis set is chosen as the basis of the orbitals, $b_j(\mathbf{r})$ for $j = \{1...n_b\}$ where n_b is the number of basis elements. As the basis set is non-orthogonal, Eq. (1) becomes the Roothaan's equation [28],

$$\mathbf{F}[n]\mathbf{p}_i = \varepsilon_i \mathbf{S}\mathbf{p}_i, \tag{5}$$

where $\mathbf{p}_i$ is the basis coefficient for the i-th orbital, $\mathbf{S}$ is the overlap matrix with elements $S_{rc} = \int b_r(\mathbf{r}) b_c^*(\mathbf{r}) \mathrm{d}\mathbf{r}$, and $\mathbf{F}[n]$ is the Fock matrix as a functional of the electron density profile, n, with elements $F_{rc} = \int b_r(\mathbf{r}) H[n](\mathbf{r}) b_c^*(\mathbf{r}) \mathrm{d}\mathbf{r}$. Equation (5) can be solved by performing an eigendecomposition of the Fock matrix to obtain the orbitals from the electron density profile.

The ionic Coulomb potential, Hartree potential, and the kinetic energy operators in the Fock matrix are constructed by calculating the Gaussian integrals with libcint [29]. The contribution from the xc potential operator in the Fock matrix is constructed by integrating it with the basis using SG-3 integration grid [30] with Becke scheme [31] to combine multi-atomic grids.

C Gradient of KS-DFT components

C.1 Self-consistent iteration cycle

The self-consistent iteration cycle in a KS-DFT calculation can be written mathematically as

$$\mathbf{y} = \mathbf{f}(\mathbf{y}, \theta), \tag{6}$$

where $\mathbf{y}$ are the parameters to be found self-consistently, and $\mathbf{f}(\cdot)$ is the function that needs to be satisfied which may depend additionally on other parameters, here denoted by θ. The self-consistent iteration takes the parameters θ and an initial guess $\mathbf{y_0}$, and produces the self-consistent parameters $\mathbf{y}$ that satisfy the equation above. In our case, the self-consistent parameters are the elements of the Fock matrix.

The gradient of a loss function $\mathcal{L}$, with respect to the parameters θ, given the gradient with respect to the self-consistent parameters $\partial\mathcal{L}/\partial\mathbf{y}$, can be expressed as

$$\frac{\partial \mathcal{L}}{\partial \theta} = -\frac{\partial \mathcal{L}}{\partial \mathbf{y}} \left(\mathbf{I} - \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right)^{-1} \left(\frac{\partial \mathbf{f}}{\partial \theta} \right), \tag{7}$$

where $\mathbf{I}$ is the identity matrix. The inverse-matrix vector calculation in the equation above is performed using the BiCGSTAB algorithm [32].

C.2 Gaussian basis

The differentiable KS-DFT employs a contracted Gaussian basis. The calculation of the Gaussian basis in 3D space is performed using libcgto, a library within PySCF [33]. Libcgto is a library that can generate code to compute a Gaussian basis of any order. As libcgto does not provide an automatic differentiation feature, it has to be wrapped by PyTorch to enable the automatic differentiation mode through the integrals calculated by libcgto.

A single Gaussian basis centered at $\mathbf{r_0}$ of order l, m, n is expressed as

$$g_{lmn}(\mathbf{r}; \alpha, c, \mathbf{r_0}) = c(x - x_0)^l (y - y_0)^m (z - z_0)^n e^{-\alpha ||\mathbf{r} - \mathbf{r_0}||^2}, \tag{8}$$

where c is the basis coefficient and α is the exponential parameter. The contracted Gaussian basis is just a linear combination of the single Gaussian bases above.

The gradient back-propagation calculation with respect to the parameters, (c, α, and $\mathbf{r_0}$) can be expressed as

$$\frac{\partial \mathcal{L}}{\partial c} = \frac{\partial \mathcal{L}}{\partial g_{lmn}} \frac{\partial g_{lmn}}{\partial c} \tag{9}$$

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \frac{\partial \mathcal{L}}{\partial g_{lmn}} \frac{\partial g_{lmn}}{\partial \alpha} \tag{10}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{r_0}} = \frac{\partial \mathcal{L}}{\partial g_{lmn}} \frac{\partial g_{lmn}}{\partial \mathbf{r_0}}, \tag{11}$$

where

$$\frac{\partial g_{lmn}}{\partial c} = \frac{g_{lmn}}{c} \tag{12}$$

$$\frac{\partial g_{lmn}}{\partial \alpha} = -\left(g_{(l+2)mn} + g_{l(m+2)n} + g_{lm(n+2)}\right) \tag{13}$$

$$\frac{\partial g_{lmn}}{\partial \mathbf{r_0}} = -\nabla g_{lmn}. \tag{14}$$

The parameters required for the calculations above (e.g. the gradient of the basis and higher order bases) can be provided by libcgto.

C.3 Gaussian integrals

Constructing the Fock matrix and the overlap matrix in Roothaan's equation requires the evaluation of Gaussian integrals. The integrals typically involves 2-4 Gaussian bases of any order. Those integrals are:

$$S_{ijk-lmn} = \int g_{ijk}(\mathbf{r}) g_{lmn}(\mathbf{r}) \, \mathrm{d}\mathbf{r} \tag{15}$$

$$K_{ijk-lmn} = -\frac{1}{2} \int g_{ijk}(\mathbf{r}) \nabla^2 g_{lmn}(\mathbf{r}) \, \mathrm{d}\mathbf{r} \tag{16}$$

$$C_{ijk-lmn} = \sum_c \int g_{ijk}(\mathbf{r}) \frac{Z_c}{|\mathbf{r} - \mathbf{r_c}|} g_{lmn}(\mathbf{r}) \, \mathrm{d}\mathbf{r} \tag{17}$$

$$E_{ijk-lmn-pqr-stu} = \int g_{ijk}(\mathbf{r}) g_{lmn}(\mathbf{r}) \frac{1}{|\mathbf{r} - \mathbf{r}'|} g_{pqr}(\mathbf{r}') g_{stu}(\mathbf{r}') \, \mathrm{d}\mathbf{r}\mathrm{d}\mathbf{r}', \tag{18}$$

where S is the overlap term, K is the kinetic term, C is the ionic Coulomb term of an ion of charge Z_c at coordinate $\mathbf{r_c}$, and E is the two-electron Coulomb term (i.e. the Hartree potential operator). These integrals can be calculated efficiently by libcint [29].

The gradients of a loss function $\mathcal{L}$ with respect to the basis parameters $\theta = \{c, \alpha, \mathbf{r_0}\}$ are:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \left(\frac{\partial \mathcal{L}}{\partial \theta}\right)_S + \left(\frac{\partial \mathcal{L}}{\partial \theta}\right)_K + \left(\frac{\partial \mathcal{L}}{\partial \theta}\right)_C + \left(\frac{\partial \mathcal{L}}{\partial \theta}\right)_E \tag{19}$$

$$\left(\frac{\partial \mathcal{L}}{\partial \theta}\right)_S = \frac{\partial \mathcal{L}}{\partial S}\left[\int \frac{\partial g_{ijk}(\mathbf{r})}{\partial \theta} g_{lmn}(\mathbf{r})\, \mathrm{d}\mathbf{r} + \int g_{ijk} \frac{\partial g_{lmn}(\mathbf{r})}{\partial \theta}\, \mathrm{d}\mathbf{r}\right] \tag{20}$$

$$\left(\frac{\partial \mathcal{L}}{\partial \theta}\right)_K = -\frac{1}{2}\frac{\partial \mathcal{L}}{\partial K}\left[\int \frac{\partial g_{ijk}(\mathbf{r})}{\partial \theta} \nabla^2 g_{lmn}(\mathbf{r})\, \mathrm{d}\mathbf{r} + \int g_{ijk} \nabla^2 \frac{\partial g_{lmn}(\mathbf{r})}{\partial \theta}\, \mathrm{d}\mathbf{r}\right] \tag{21}$$

$$\left(\frac{\partial \mathcal{L}}{\partial \theta}\right)_C = \frac{\partial \mathcal{L}}{\partial C}\sum_c \left[\int \frac{\partial g_{ijk}(\mathbf{r})}{\partial \theta} \frac{Z_c}{|\mathbf{r} - \mathbf{r_c}|} g_{lmn}(\mathbf{r})\, \mathrm{d}\mathbf{r} + \int g_{ijk} \frac{Z_c}{|\mathbf{r} - \mathbf{r_c}|} \frac{\partial g_{lmn}(\mathbf{r})}{\partial \theta}\, \mathrm{d}\mathbf{r}\right] \tag{22}$$

$$\begin{aligned}\left(\frac{\partial \mathcal{L}}{\partial \theta}\right)_E = \frac{\partial \mathcal{L}}{\partial E}\Bigg[&\int \frac{\partial g_{ijk}(\mathbf{r})}{\partial \theta} \frac{g_{lmn}(\mathbf{r}) g_{pqr}(\mathbf{r}') g_{stu}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}\, \mathrm{d}\mathbf{r} + \int \frac{\partial g_{lmn}(\mathbf{r})}{\partial \theta} \frac{g_{ijk}(\mathbf{r}) g_{pqr}(\mathbf{r}') g_{stu}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}\, \mathrm{d}\mathbf{r} \\ &+ \int \frac{\partial g_{pqr}(\mathbf{r}')}{\partial \theta} \frac{g_{ijk}(\mathbf{r}) g_{lmn}(\mathbf{r}) g_{stu}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}\, \mathrm{d}\mathbf{r} + \int \frac{\partial g_{stu}(\mathbf{r}')}{\partial \theta} \frac{g_{ijk}(\mathbf{r}) g_{lmn}(\mathbf{r}) g_{pqr}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}\, \mathrm{d}\mathbf{r}\Bigg].\end{aligned} \tag{23}$$

If the parameter θ does not belong to the basis $g.$, then $\partial g./\partial\theta$ is 0. Otherwise, the values of $\partial g./\partial\theta$ for various parameters are listed in the previous subsection. For the gradient with respect to the atomic position, $\mathbf{r_c}$, the expression is given by

$$\frac{\partial \mathcal{L}}{\partial \mathbf{r_c}} = \frac{\partial \mathcal{L}}{\partial C}\left[\int \nabla g_{ijk}(\mathbf{r}) \frac{Z_c}{|\mathbf{r} - \mathbf{r_c}|} g_{lmn}(\mathbf{r})\, \mathrm{d}\mathbf{r} + \int g_{ijk}(\mathbf{r}) \frac{Z_c}{|\mathbf{r} - \mathbf{r_c}|} \nabla g_{lmn}(\mathbf{r})\, \mathrm{d}\mathbf{r}\right]. \tag{24}$$

The expression above is obtained by performing partial integration. As the derivative of the Gaussian basis with respect to its parameters is also a Gaussian basis, the required integrals can be provided by libcint [29].

C.4 Exchange-correlation energy and potential

The exchange-correlation (xc) energy in our calculation is a hybrid between available xc energy functionals (i.e. the LDA [11] and PBE [20] models) and a neural network xc energy. The gradient back-propagation through the xc neural network is easily provided by PyTorch. However, as we use Libxc [21] to calculate the established xc energy, it has to be wrapped by PyTorch to provide the gradient back-propagation calculation. As Libxc can also calculate the gradient of the energy with respect to its parameters, providing the automatic differentiation feature by wrapping it with PyTorch is just a matter of calling the right function in Libxc.

If the PyTorch wrapper for Libxc is provided, calculating the xc potential can be done using the automatic differentiation feature provided by PyTorch. The component of the Fock matrix from the xc potential for the LDA spin-polarized case is

$$V_{ij}^{(\mathrm{LDA})} = \int \phi_i^*(\mathbf{r}) v_s^{(\mathrm{LDA})}(\mathbf{r}) \phi_j(\mathbf{r})\, \mathrm{d}\mathbf{r}, \tag{25}$$

where the subscript s can be $\uparrow$ or $\downarrow$ representing the spin of the electron, and $v_s^{(\mathrm{LDA})}$ is the xc potential,

$$v_s^{(\mathrm{LDA})} = \frac{\partial}{\partial n_s}\int n\varepsilon^{(\mathrm{LDA})}(n_\uparrow, n_\downarrow)\, \mathrm{d}\mathbf{r}, \tag{26}$$

with $\varepsilon^{(\mathrm{LDA})}$ the energy density per unit volume, per electron, provided by Libxc. The 3D spatial integration is performed using SG-3 integration grid [30] with Becke scheme [31] to combine multi-atomic grids. The derivation with respect to n_s in the equation above can simply be calculated using the automatic differentiation feature of PyTorch wrapping the Libxc library.

For spin-polarized GGA, the potential linear operator is [34]

$$V_{ij}^{(\mathrm{GGA})} = \int \phi_i^*(\mathbf{r}) v_s^{(\mathrm{GGA})}(\mathbf{r}) \phi_j(\mathbf{r})\, \mathrm{d}\mathbf{r} \tag{27}$$

$$v_{\mathrm{GGAs}} = \frac{\partial}{\partial n_s}\int n\varepsilon^{(\mathrm{GGA})}(n_\uparrow, n_\downarrow, \nabla n_\uparrow, \nabla n_\downarrow)\, \mathrm{d}\mathbf{r} + \left(\frac{\partial}{\partial \nabla n_s}\int n\varepsilon^{(\mathrm{GGA})}\, \mathrm{d}\mathbf{r}\right)\cdot\nabla, \tag{28}$$

where the derivatives with respect to n_s and ∇n_s can be performed by automatic differentiation, and the last ∇ term is applied to the basis which can be provided by libcgto from PySCF [33].

C.5 (Near-)degenerate case of eigendecomposition

One challenge in propagating the gradient backward for some molecules is the degeneracy of eigenvalues in the eigendecomposition. The eigendecomposition of a real and symmetric matrix, $\mathbf{A}$, can be written as

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{29}$$

with λ_i and $\mathbf{v}_i$ as the i-th eigenvalue and eigenvector, respectively.

In the non-degenerate case, the gradient of a loss function $\mathcal{L}$ with respect to each element of matrix $\mathbf{A}$ can be written as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = -\sum_j \sum_{i \neq j} (\lambda_i - \lambda_j)^{-1} \mathbf{v}_i \mathbf{v}_i^T \frac{\partial \mathcal{L}}{\partial \mathbf{v}_j} \mathbf{v}_j^T. \tag{30}$$

A problem arises when degeneracy appears as a term of 0^{-1} appears in the equation above.

To solve this problem, we follow the equation from ref [18], which can be written simply as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = -\sum_j \sum_{i \neq j, \lambda_i \neq \lambda_j} (\lambda_i - \lambda_j)^{-1} \mathbf{v}_i \mathbf{v}_i^T \frac{\partial \mathcal{L}}{\partial \mathbf{v}_j} \mathbf{v}_j^T, \tag{31}$$

which just removes the degenerate terms in the sum. However, the equation above is only valid as long as the loss function does not depend on which linear combinations of the degenerate eigenvectors are used and the elements in matrix $\mathbf{A}$ depends on some parameters that always make the matrix $\mathbf{A}$ real and symmetric. Otherwise, the equation above becomes invalid and the actual gradient becomes mathematically undefined.

D Neural network training

D.1 Dataset

The atomization energy dataset is compiled from NIST CCCBDB [22]. CCCBDB contains the experimental data for enthalpy of atomization energy. To convert it into electronic atomization energy which can be calculated by KS-DFT calculations, the zero-point vibration energy must be excluded from the enthalpy of atomization energy. Some diatomic molecules have the experimental values of the zero-point energy in CCCBDB from ref [35]. For the rest of the molecules, the zero-point energy is estimated from the fundamental vibrations of the molecules which are also available from CCCBDB.

D.2 Neural networks

The neural network is an ordinary feed-forward neural network with 3 hidden layers consist of 32 elements each with softplus [36] activation function to have infinite differentiability of the neural network. To avoid non-convergence of the self-consistent iterations during the training, the parameters α and β are initialized to have values of 1 and 0 respectively.

The neural network takes an input of the density, n, spin polarized density, $\xi = (n_\uparrow - n_\downarrow)/n$, and normalized density gradient, $s = |\nabla n|^2/(24\pi^2 n^4)^{1/3}$ (for XCNN-PBE). As the density (n) and the normalized density gradient (s) can have values between 0 up to a very large number, these values are renormalized by applying $\log(1+x)$, i.e. $n \leftarrow \log(1+n)$ and $s \leftarrow \log(1+s)$.

D.3 Training

The parameters of XCNN are updated based on the gradient of a loss function with respect to its parameters. The loss function is given by a combination of two terms

$$\mathcal{L} = \frac{w_{ae}}{N_{ae}} \sum_{i=1}^{N_{ae}} \left(\hat{E}_i^{(ae)} - E_i^{(ae)} \right)^2 + \frac{w_{dp}}{N_{dp}} \sum_{i=1}^{N_{dp}} \int \left[\hat{n}_i(\mathbf{r}) - n_i(\mathbf{r}) \right]^2 \mathrm{d}\mathbf{r}, \tag{32}$$

where $w_{(\cdot)}$ are the weights assigned to each type of the data, $N_{(\cdot)}$ are the number of entries of each datatype in the dataset, $E^{(ae)}$ and $E^{(ip)}$ are, respectively, the predicted atomization energy and

ionization potential using XCNN and KS-DFT, n is the electron density profile predicted with XCNN and KS-DFT. The variables with a hat, $\hat{E}^{(\cdot)}$ and $\hat{n}$, are the target variables that were obtained from experimental databases (for energies) and CCSD calculations (for electron density profiles).

The XCNN is trained using the RAdam [37] algorithm with a constant learning rate of 10^{-4}. The algorithm and learning rate were chosen arbitrarily without further parameter tuning due to resource and time limitations. The weights for training of the neural networks are:

- XCNN-LDA: $w_{ae} = 1340$, $w_{dp} = 170$.
- XCNN-PBE: $w_{ae} = 1340$, $w_{dp} = 5360$.

The weights were chosen to make the mean error for each component approximately equals to 1.

The XCNN training took 2 weeks using 32 CPU cores. We did not use GPU in this training because the integral calculations in DQC are not written for GPU.

E Molecules in the test dataset

The 104 molecules in the test dataset, as well as the subset they belong, to are listed below.

1. LiH (Lithium Hydride): NHC-1
2. BeH (beryllium monohydride): NHC-1
3. CH (Methylidyne): HC
4. CH_3 (Methyl radical): HC
5. CH_4 (Methane): HC
6. NH (Imidogen): NHC-1
7. NH_2 (Amino radical): NHC-1
8. NH_3 (Ammonia): NHC-1
9. OH (Hydroxyl radical): NHC-1
10. H_2O (Water): NHC-1
11. HF (Hydrogen fluoride): NHC-1
12. SiH_2 (silicon dihydride): NHC-2
13. SiH_3 (Silyl radical): NHC-2
14. SiH_4 (Silane): NHC-2
15. PH_3 (Phosphine): NHC-2
16. H_2S (Hydrogen sulfide): NHC-2
17. HCl (Hydrogen chloride): NHC-2
18. Li_2 (Lithium diatomic): NHC-1
19. LiF (lithium fluoride): NHC-1
20. C_2H_2 (Acetylene): HC
21. C_2H_4 (Ethylene): HC
22. C_2H_6 (Ethane): HC
23. CN (Cyano radical): NHC-1
24. HCN (Hydrogen cyanide): NHC-1
25. CO (Carbon monoxide): NHC-1
26. HCO (Formyl radical): NHC-1
27. H_2CO (Formaldehyde): NHC-1
28. CH_3OH (Methyl alcohol): SHC
29. N_2 (Nitrogen diatomic): NHC-1

30. N_2H_4 (Hydrazine): NHC-1
31. NO (Nitric oxide): NHC-1
32. O_2 (Oxygen diatomic): NHC-1
33. H_2O_2 (Hydrogen peroxide): NHC-1
34. F_2 (Fluorine diatomic): NHC-1
35. CO_2 (Carbon dioxide): NHC-1
36. Na_2 (Disodium): NHC-2
37. Si_2 (Silicon diatomic): NHC-2
38. P_2 (Phosphorus diatomic): NHC-2
39. S_2 (Sulfur diatomic): NHC-2
40. Cl_2 (Chlorine diatomic): NHC-2
41. NaCl (Sodium Chloride): NHC-2
42. SiO (Silicon monoxide): NHC-2
43. CS (carbon monosulfide): NHC-2
44. SO (Sulfur monoxide): NHC-2
45. ClO (Monochlorine monoxide): NHC-2
46. ClF (Chlorine monofluoride): NHC-2
47. Si_2H_6 (disilane): NHC-2
48. CH_3Cl (Methyl chloride): SHC
49. HOCl (hypochlorous acid): NHC-2
50. SO_2 (Sulfur dioxide): NHC-2
51. BF_3 (Borane trifluoro-): NHC-1
52. BCl_3 (Borane trichloro-): NHC-2
53. AlF_3 (Aluminum trifluoride): NHC-2
54. $AlCl_3$ (Aluminum trichloride): NHC-2
55. CF_4 (Carbon tetrafluoride): NHC-1
56. CCl_4 (Carbon tetrachloride): NHC-2
57. OCS (Carbonyl sulfide): NHC-2
58. CS_2 (Carbon disulfide): NHC-2
59. CF_2O (Carbonic difluoride): NHC-1
60. SiF_4 (Silicon tetrafluoride): NHC-2
61. $SiCl_4$ (Silane tetrachloro-): NHC-2
62. N_2O (Nitrous oxide): NHC-1
63. ClNO (Nitrosyl chloride): NHC-2
64. NF_3 (Nitrogen trifluoride): NHC-1
65. O_3 (Ozone): NHC-1
66. F_2O (Difluorine monoxide): NHC-1
67. C_2F_4 (Tetrafluoroethylene): NHC-1
68. CF_3CN (Acetonitrile trifluoro-): NHC-1
69. CH_3CCH (propyne): HC
70. CH_2CCH_2 (allene): HC
71. C_3H_4 (cyclopropene): HC
72. C_3H_6 (Cyclopropane): HC
73. C_3H_8 (Propane): HC

74. C_4H_6 (Methylenecyclopropane): HC
75. C_4H_6 (Cyclobutene): HC
76. $CH_3CH(CH_3)CH_3$ (Isobutane): HC
77. C_6H_6 (Benzene): HC
78. CH_2F_2 (Methane difluoro-): SHC
79. CHF_3 (Methane trifluoro-): SHC
80. CH_2Cl_2 (Methylene chloride): SHC
81. $CHCl_3$ (Chloroform): SHC
82. CH_3CN (Acetonitrile): SHC
83. HCOOH (Formic acid): SHC
84. CH_3CONH_2 (Acetamide): SHC
85. C_2H_5N (Aziridine): SHC
86. C_2N_2 (Cyanogen): SHC
87. CH_2CO (Ketene): SHC
88. C_2H_4O (Ethylene oxide): SHC
89. $C_2H_2O_2$ (Ethanedial): SHC
90. CH_3CH_2OH (Ethanol): SHC
91. CH_3OCH_3 (Dimethyl ether): SHC
92. C_2H_4O (Ethylene oxide): SHC
93. CH_2CHF (Ethene fluoro-): SHC
94. CH_3CH_2Cl (Ethyl chloride): SHC
95. CH_3COF (Acetyl fluoride): SHC
96. CH_3COCl (Acetyl Chloride): SHC
97. C_4H_4O (Furan): SHC
98. C_4H_5N (Pyrrole): SHC
99. H_2 (Hydrogen diatomic): NHC-1
100. HS (Mercapto radical): NHC-2
101. C_2H (Ethynyl radical): HC
102. CH_3O (Methoxy radical): SHC
103. CH_3S (thiomethoxy): SHC
104. NO_2 (Nitrogen dioxide): NHC-1