Neural network distillation of orbital dependent density functional theory

Supplementary Material

Matija Medvidović,[1] Jaylyn C. Umana,[2,3,4] Iman Ahmadabadi,[5,2,6]
Domenico Di Sante,[7] Johannes Flick,[3,4,2] and Angel Rubio[8,2,9]

[1] *Institute for Theoretical Physics, ETH Zürich, CH-8093 Zürich, Switzerland*
[2] *Center for Computational Quantum Physics, Flatiron Institute, 162 5th Avenue, New York, NY 10010, USA*
[3] *Department of Physics, City College of New York, New York, NY 10031, USA*
[4] *Department of Physics, The Graduate Center, City University of New York, New York, NY 10016, USA*
[5] *Joint Quantum Institute, NIST and University of Maryland, College Park, MD 20742, USA*
[6] *Department of Chemistry, Princeton University, Princeton, NJ 08544, USA*
[7] *Department of Physics and Astronomy, University of Bologna, 40127 Bologna, Italy*
[8] *Max Planck Institute for the Structure and Dynamics of Matter, Luruper Chaussee 149, 22761 Hamburg, Germany*
[9] *Initiative for Computational Catalysis, Flatiron Institute, 162 5th Avenue, New York, NY 10010, USA*

(Dated: April 13, 2025)

I. NEURAL NETWORK ARCHITECTURE

A. Linear attention layers as continuous convolutions

We consider input data in a point cloud format of density values with coordinates $\mathbf{r}_i$, density values $n_i = n(\mathbf{r}_i)$ and quadrature weights w_i where $i = 1, \ldots, N_g$ indexes the cloud in arbitrary order. Functional inputs in this form are general and can be readily extracted from modern quantum chemistry software packages. This enables us to approximate integrals of sufficiently well-behaved functions as:

$$\int \mathrm{d}^3\mathbf{r}\, f(\mathbf{r}) \approx \sum_i w_i\, f(\mathbf{r}_i)\ . \tag{1}$$

Global density approximations rely on the linear attention [1] mechanism for propagating information between local density values. In purely mathematical terms, the linear self-attention operation used in this work can be seen as a discretization of an integral transform with a learnable kernel $\mathcal{K}$:

$$\phi'(\mathbf{r}) = \int \mathrm{d}^3\mathbf{r}'\, n(\mathbf{r}')\mathcal{K}(\mathbf{r},\mathbf{r}')\phi(\mathbf{r}') = \int \mathrm{d}^3\mathbf{r}'\, n(\mathbf{r}')\left(Q(\mathbf{r})\cdot K(\mathbf{r}')\right)V(\mathbf{r}') \tag{2}$$

where $\mathcal{K}$ has been parameterized in factorized form in order to exploit the *kernel trick*.

In Eq. 2, we define the so-called queries, keys, and values $Q, K, V \in \mathbb{R}^d$ as learnable local linear transformations of the input field $\phi(\mathbf{r}) \in \mathbb{R}^d$:

$$Q(\mathbf{r}) = W^Q\phi(\mathbf{r})\,,\ K(\mathbf{r}) = W^K\phi(\mathbf{r}) \quad \text{and} \quad V(\mathbf{r}) = W^V\phi(\mathbf{r})\ . \tag{3}$$

The functional form of Eq. 2 should be contrasted with an equivalent definition of the standard `softmax` nonlinear attention [2, 3] kernel head

$$\mathcal{K}_{\mathrm{sm}}(\mathbf{r},\mathbf{r}') = \frac{e^{Q(\mathbf{r})\cdot K(\mathbf{r}')}}{\int \mathrm{d}^3\mathbf{r}'\, e^{Q(\mathbf{r})\cdot K(\mathbf{r}')}} \tag{4}$$

where a similar kernel trick cannot be exploited.

Large integration grids of $N_g > 10^6$ points are not uncommon in modern DFT calculations. Therefore, the naive `softmax` self-attention operation between all grid points with asymptotic scaling $\mathcal{O}(N_g^2)$ is prohibitively numerically expensive. For large point cloud datasets like densities $n(\mathbf{r})$ on Becke grids, this is necessary to avoid materializing the full $N_g \times N_g$ kernel matrix $\mathcal{K}(\mathbf{r}_i, \mathbf{r}_j)$, incurring prohibitively large memory requirements. Instead, in Eq. 2, we first contract $K(\mathbf{r})$ and $V(\mathbf{r}')$, then perform the integral and, finally, multiply by $Q(\mathbf{r})$.

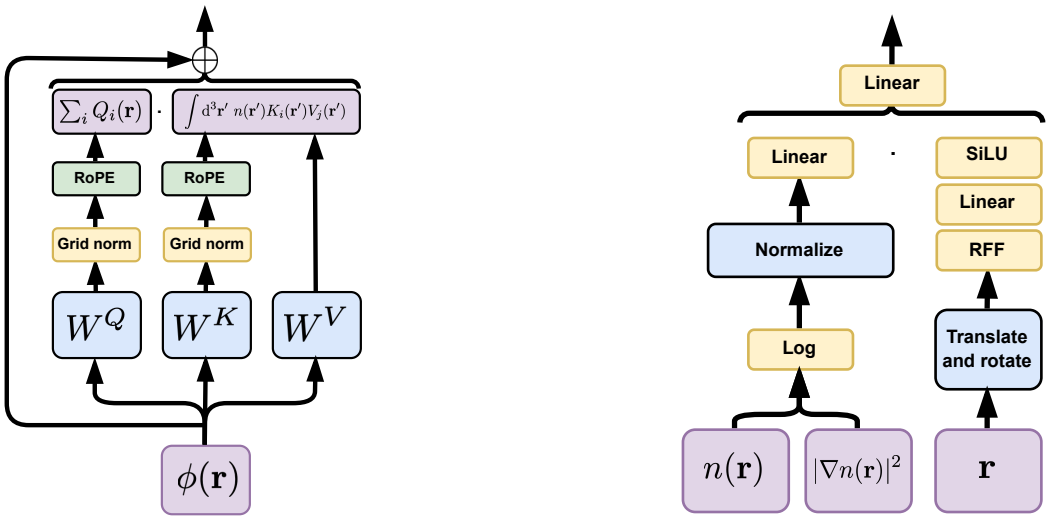

FIG. 1. Diagrammatic representations of internal connectivities of GDA sub-components. **Left**: The linear attention module enabling the nonlocal behavior of resulting functionals. **Right**: The field embedding module lifting the local field representations to a high-dimentional space.

Additionally, we apply custom *grid normalization*, to field projections of the input field ϕ defined in Eq. 3 before performing the attention operation in Eq. 2. Grid normalization amounts to normalizing keys K and queries Q in the following way:

$$f(\mathbf{r}) \;\mapsto\; \frac{f(\mathbf{r})}{\sqrt{\int \mathrm{d}^3\mathbf{r}\, n(\mathbf{r})(f(\mathbf{r}))^2}}\,, \tag{5}$$

in an component-wise way for multi-component fields. We note that the choice of leaving values V unnormalized has been called *Fourier* linear attention as opposed to *Galerkin* linear attention where keys and values are normalized.

Furthermore, in order to ensure that the resulting kernel $\mathcal{K}(\mathbf{r}-\mathbf{r}') = Q(\mathbf{r})\cdot K(\mathbf{r})$ is purely a function of the relative coordinate, we employ rotary positional encoding (RoPE) [4, 5]. The RoPE mechanism relies on the composition propery of representations D of the rotation group $SO(2)$ in two dimensions: $D(\alpha)D(\alpha') = D(\alpha+\alpha')$ for angles α and α'.

We generalize the RoPE mechanism to three spatial dimensions by setting $\alpha = \mathbf{k}\cdot\mathbf{r}$ for an input at point $\mathbf{r}\in\mathbb{R}^3$ and include $\mathbf{k}\in\mathbb{R}^3$ in the learnable parameters of the model. Therefore, we have

$$D(\mathbf{k}\cdot\mathbf{r})D(-\mathbf{k}\cdot\mathbf{r}') = D(\mathbf{k}\cdot\mathbf{r})D^{-1}(\mathbf{k}\cdot\mathbf{r}') = D(\mathbf{k}\cdot(\mathbf{r}-\mathbf{r}'))\,. \tag{6}$$

To construct an orthogonal representation D operating on the input field components $\phi_i(\mathbf{r})$ and exploit Eq. 6, we first note that a trivial representation of SU(2) on $\mathbb{R}^2$ is given by simple rotation matrices

$$D_{\mathbb{R}^2}(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}. \tag{7}$$

Therefore, a multi-component input field $\phi(\mathbf{r})$ with an even number of components, we define the 3D RoPE as a trivially reducible orthogonal representation $\phi(\mathbf{r}) \mapsto D_{\mathbb{R}^{2n}}(\mathbf{k}\cdot\mathbf{r})\,\phi(\mathbf{r})$ with

$$D_{\mathbb{R}^{2n}}(\mathbf{k}_1\cdot\mathbf{r},\ldots,\mathbf{k}_n\cdot\mathbf{r}) = \begin{bmatrix} \cos(\mathbf{k}_1\cdot\mathbf{r}) & \sin(\mathbf{k}_1\cdot\mathbf{r}) & & & \\ -\sin(\mathbf{k}_1\cdot\mathbf{r}) & \cos(\mathbf{k}_1\cdot\mathbf{r}) & & & \\ & & \ddots & & \\ & & & \cos(\mathbf{k}_n\cdot\mathbf{r}) & \sin(\mathbf{k}_n\cdot\mathbf{r}) \\ & & & -\sin(\mathbf{k}_n\cdot\mathbf{r}) & \cos(\mathbf{k}_n\cdot\mathbf{r}) \end{bmatrix} \tag{8}$$

depending on n rotation angles defined by trainable parameters $\mathbf{K} = \{\mathbf{k}_1,\ldots,\mathbf{k}_n\}$. The block-diagonal structure in Eq. 8 mixes adjacent components of the input field in a way that reflects the target $SO(2)$ multiplication property.

In that case, the query-key product in Eq. 2

$$\begin{aligned}\mathcal{K}(\mathbf{r},\mathbf{r}') =& Q(\mathbf{r})\cdot K(\mathbf{r}') = \\ =& [D(\mathbf{K}\cdot\mathbf{r})W^Q\phi(\mathbf{r})]^\top [D(\mathbf{K}\cdot\mathbf{r}')W^K\phi(\mathbf{r})] = \\ =& \phi(\mathbf{r})^\top (W^Q)^\top D^\top(\mathbf{K}\cdot\mathbf{r})D(\mathbf{K}\cdot\mathbf{r}')W^K\phi(\mathbf{r}) = \\ =& \phi(\mathbf{r})^\top (W^Q)^\top D^\top(-\mathbf{K}\cdot(\mathbf{r}-\mathbf{r}'))W^K\phi(\mathbf{r}) = \\ =& \mathcal{K}(\mathbf{r}-\mathbf{r}')\end{aligned}$$

reduces the linear attention integral to a learnable continuous convolution, if we apply the 3D RoPE transformation as a final step just before integration.

Finally, after the attention layer has been updated, the output field ϕ is transformed by a component-wise gated multi-layer perceptron [6]:

$$\phi(\mathbf{r}) \;\mapsto\; W_3\left((W_1\phi(\mathbf{r})+\mathbf{b}_1)\odot\sigma(W_2\phi(\mathbf{r})+\mathbf{b}_2)\right)+\mathbf{b}_3 \tag{9}$$

with `SiLU` activations [7] $\sigma(x)=\frac{x}{1+e^{-x}}$. Weights W_i and biases $\mathbf{b}_i$ are included in trainable parameters and $\odot$ indicates element-wise multiplication.

In summary, the steps comprising one GDA *block* are:

1. Evaluate raw queries, keys and values by linear projections in Eq. 3.
2. Normalize queries and keys using grid normalization in Eq. 5.
3. Apply the 3D RoPE transformation in Eq. 8 to queries and keys.
4. Evaluate the final linear attention integral in Eq. 2 using the kernel trick. Add a skip connection.
5. Normalize [8] and apply the gated MLP in Eq. 9. Add a skip connection.

Internal connectivity of sub-layers can be found in Fig. 1.

B. Field embedding

The field embedding layer is used in the GDA architecture to ensure correct symmetry properties of the learned model. Since the model is explicitly coordinate-dependent, we fix a coordinate system to perform the computation with comparable values of resulting *computational* coordinates. We choose the coordinate system in which the mean value (electronic dipole moment)

$$\boldsymbol{\mu} = \frac{1}{N}\int \mathrm{d}^3\mathbf{r}\, n(\mathbf{r})\,\mathbf{r} \tag{10}$$

vanishes and the covariance matrix

$$\Sigma_{ij} = \frac{1}{N}\int \mathrm{d}^3\mathbf{r}\, n(\mathbf{r})\,(r_i-\mu_i)\,(r_j-\mu_j) \tag{11}$$

is diagonal. After fixing translations and rotations, individual coordinates $\mathbf{r}_i$ are independently embedded into a d-dimensional space as

$$\mathbf{r}\mapsto\xi(\mathbf{r}) = \frac{1}{\sqrt{d}}\left[\cos\left(K_0\mathbf{r}\right)\;\|\;\sin\left(K_0\mathbf{r}\right)\right]^\top, \tag{12}$$

where $\|$ denotes vector concatenation and $K_0\in\mathbb{R}^{\frac{d}{2}\times 3}$ is a trainable matrix randomly initialized with values sampled from the normal distribution $\mathcal{N}(0,\sigma^{-2})$. This *random Fourier feature* (RFF) encoding has effectively been used to encode high-frequency maps of continuous coordinates in kernel learning [12], and has been shown to measurably increase expressivity of various architectures in geometric deep learning [13–15], including in combination with transformers [16].

The mapping $\mathbf{r}\mapsto\xi(\mathbf{r})$ in Eq. 12 is designed to take advantage of the (approximate) kernel trick described in the main text, in a related way to to the RoPE mechanism described in Eq. 8. However, we emphasize that it is often used outside the kernel context, as a featurization layer (called *positional encoding*) for coordinate-dependent neural

Symbol	Name	Value	Domain	Description
L	Number of blocks	3	$\mathbb{N}$	GDA block count
d	Field embedding dimension	128	$\mathbb{N}$	Dimension of internal field representations within the model
σ	RFF kernel scale	1	$\mathbb{R}_+$	Gaussian scale used to initialize the RFF embedding layer
η	Learning rate	Scheduled $2 \times 10^{-4} \to 5 \times 10^{-5}$	$\mathbb{R}_+$	(R)Adam optimizer learning rate [9, 10]
B	Batch size	288	$\mathbb{N}$	Number of molecules used for cost gradient estimation
N_e	Number of epochs	4000	$\mathbb{N}$	Number iterations over the entire dataset
λ	Relative cost weight	1	$\mathbb{R}_+$	Constant mutiplier for the potential matrix penalty in the main text
$\tilde{\lambda}$	Weight decay	0.01	$\mathbb{R}_+$	Weight decay regularizer for the network parameters [11]
α	Enhancement	2	$\mathbb{R}_+$	Relative increase in the number of features in the middle MLP layer.
t_1	Learning rate annealing start	$N_e/3$	$\mathbb{N}$	Epoch at which the learning rate annealing starts (see Fig. 2)
t_2	Learning rate annealing end	$0.95\, N_e$	$\mathbb{N}$	Epoch at which the learning rate annealing ends (see Fig. 2)

TABLE I. The list of relevant hyperparameter choices used in this work.

network inputs. Suppose that a convolution integral with a kernel $\mathcal{K}$ needs evaluation. Denoting Fourier transforms with hats and employing the convolution theorem in combination with Monte Carlo integration, we have:

$$\int \mathrm{d}^3\mathbf{r}'\; \mathcal{K}(\mathbf{r}-\mathbf{r}')f(\mathbf{r}') = \int \frac{\mathrm{d}^3\mathbf{k}}{(2\pi)^3}\, \hat{\mathcal{K}}(\mathbf{k})\, \hat{f}(\mathbf{k})\, e^{i\mathbf{k}\cdot\mathbf{r}} \approx \frac{1}{(2\pi)^3} \times \frac{1}{N_s} \sum_j \hat{f}(\mathbf{k}_j)\, e^{i\mathbf{k}_j\cdot\mathbf{r}} \tag{13}$$

for N_s samples $\mathbf{k}_j \sim \hat{\mathcal{K}}$, assuming the the kernel $\mathcal{K}$ is a positive distribution on both sides of the Fourier transform. Therefore, the spread of the distribution in determines the initialization of K_0 and it is, in turn, determined by the inverse of the spread of $\mathcal{K}$ in real space, constrained by the uncertainty relation. If we choose to initialize the kernel as a Gaussian $\mathcal{N}$, we have $\mathbf{k}_j \sim \exp\left(-\frac{1}{2}\sigma^2\mathbf{k}^2\right) \propto \mathcal{N}(0, \sigma^{-2})$.

Local density field values are embedded into a d-dimensional space as well with learnable linear projections of normalized values. We take local field information in the form of $n(\mathbf{r})$ and $\gamma(\mathbf{r}) = |\nabla n(\mathbf{r})|^2$ and construct

$$\overline{\ln n(\mathbf{r})} = \frac{\ln n(\mathbf{r}) - \langle \ln n(\mathbf{r})\rangle}{\sqrt{\langle (\ln n(\mathbf{r}) - \langle \ln n(\mathbf{r})\rangle)^2\rangle}} \quad \text{and} \quad \overline{\ln \gamma(\mathbf{r})} = \frac{\ln \gamma(\mathbf{r}) - \langle \ln \gamma(\mathbf{r})\rangle}{\sqrt{\langle (\ln \gamma(\mathbf{r}) - \langle \ln \gamma(\mathbf{r})\rangle)^2\rangle}}\,, \tag{14}$$

where we use the density average $\langle\cdot\rangle = \frac{1}{N}\int \mathrm{d}^3\mathbf{r}'\, n(\mathbf{r})(\cdot)$, normalized to unity. In practice, we initialize

$$\phi(\mathbf{r}) = W^E \begin{bmatrix} \overline{\ln(n(\mathbf{r}) + \epsilon)} \\ \overline{\ln(\gamma(\mathbf{r}) + \epsilon)} \end{bmatrix} \tag{15}$$

with $\epsilon = 10^{-4}$, instead of bare logarithms for numerical stability and $W^E \in \mathbb{R}^{d\times 2}$ are trainable parameters represented by trainable unit vectors in the main text. After input fields and coordinates have been independently processed, we combine them using learnable weights W and biases b

$$\phi(\mathbf{r}) \mapsto W'(\phi(\mathbf{r}) \odot \sigma(W^\xi \xi(\mathbf{r}) + b^\xi)) + b'\,. \tag{16}$$

to produce final density embeddings. We use $\sigma = \texttt{SiLU}$ as an element-wise nonlinear activation function [7]. A list of all relevant hyperparameters used in this work can be found in table I B. A diagram of the overall connectivuty of the embedding layer can be found in the right panel of Fig. 1.

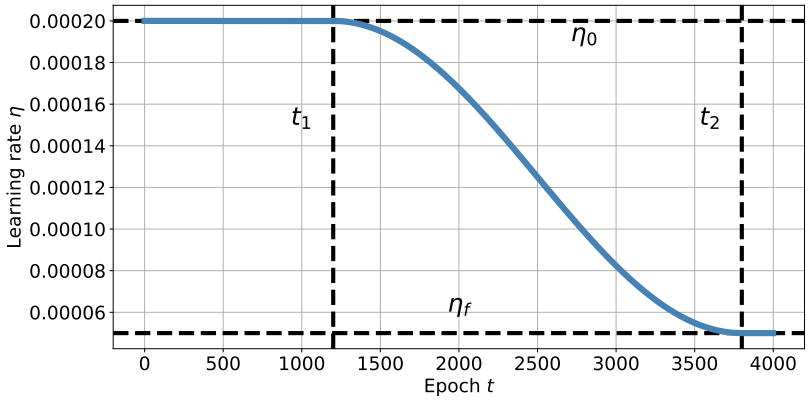

FIG. 2. The learning rate scheduling used to optimize the full GDA model.

II. OPTIMIZATION AND CONVERGENCE

A. Neural-network training

As noted in the main text, the GDA neural network was trained with $L = 3$ blocks, internal field embedding dimension $d = 128$, totaling 645000 parameters. The rectified Adam [9] (RAdam) [10] optimizer was used for $N_e = 4000$ epochs with learning rate decreasing from $\eta_i = 2 \times 10^{-4}$ to $\eta_f = 5 \times 10^{-5}$ using a customized cosine annealing schedule defined as

$$\eta_t = \begin{cases} \eta_i, & \text{if } t < t_1 \\ \eta_f + (\eta_i - \eta_f)\cos^2\left(\frac{\pi}{2}\frac{t-t_1}{t_2-t_1}\right), & \text{if } t_1 \le t \le t_2 \\ \eta_f, & \text{if } t > t_2 \end{cases} \tag{17}$$

for epoch t. The schedule is shown in Fig. 2. We find that decreasing the learning rate during the course of training helps fine-tune the model in the later epochs when the loss landscape changes at smaller scales. We set $t_1 = N_e/3$ and $t_2 = 0.95N_e$.

B. Self-consistent field optimization from first principles

We use PySCF [17, 18] for first-principles optimizations of molecular densities, where the default method is the standard SCF Pulay mixing or direct inversion in the iterative subspace (DIIS) [19]. Using that method, we optimize a random molecule from the training dataset a total of six times – three times with library [20] versions of targeted meta-GGA functionals [21–23] and three times with the GDA substitution $\tau \to \tau_\theta$. Results can be seen in the left panel of Fig. 3.

We note that r^{2}SCAN and TPSS converge in a similar number of mixing iterations to the original functionals, while SCAN (included for comparison) appears to be more sensitive to numerical errors associated with the GDA approximation. We report that this trend persists in most of the other molecules that we examined more closely.

III. NUMERICAL PERFORMANCE BENCHMARKS

In this subsection, we state some numerical heuristics when comparing the efficiency of the GDA approximation when compared to the parent meta-GGA functional. We note that, like many computational benchmarks, it is difficult to control all variables contributing to practical wall-clock times. That is especially true for the case of DFT within the context of parametrized models. Such modern AI models are accelerated using graphical processing units (GPUs), offering speedups coming massive parallelism for some operations (e.g. matrix multiplication).

Therefore, any fair comparison between efficient implementations of *traditional* and machine-learned density and orbital functionals will almost certainly be *apples-to-oranges* to some degree. Results change with different (and rapidly evolving) hardware, suffer from CPU-GPU communication overhead, just to name a few factors. Additionally, desired

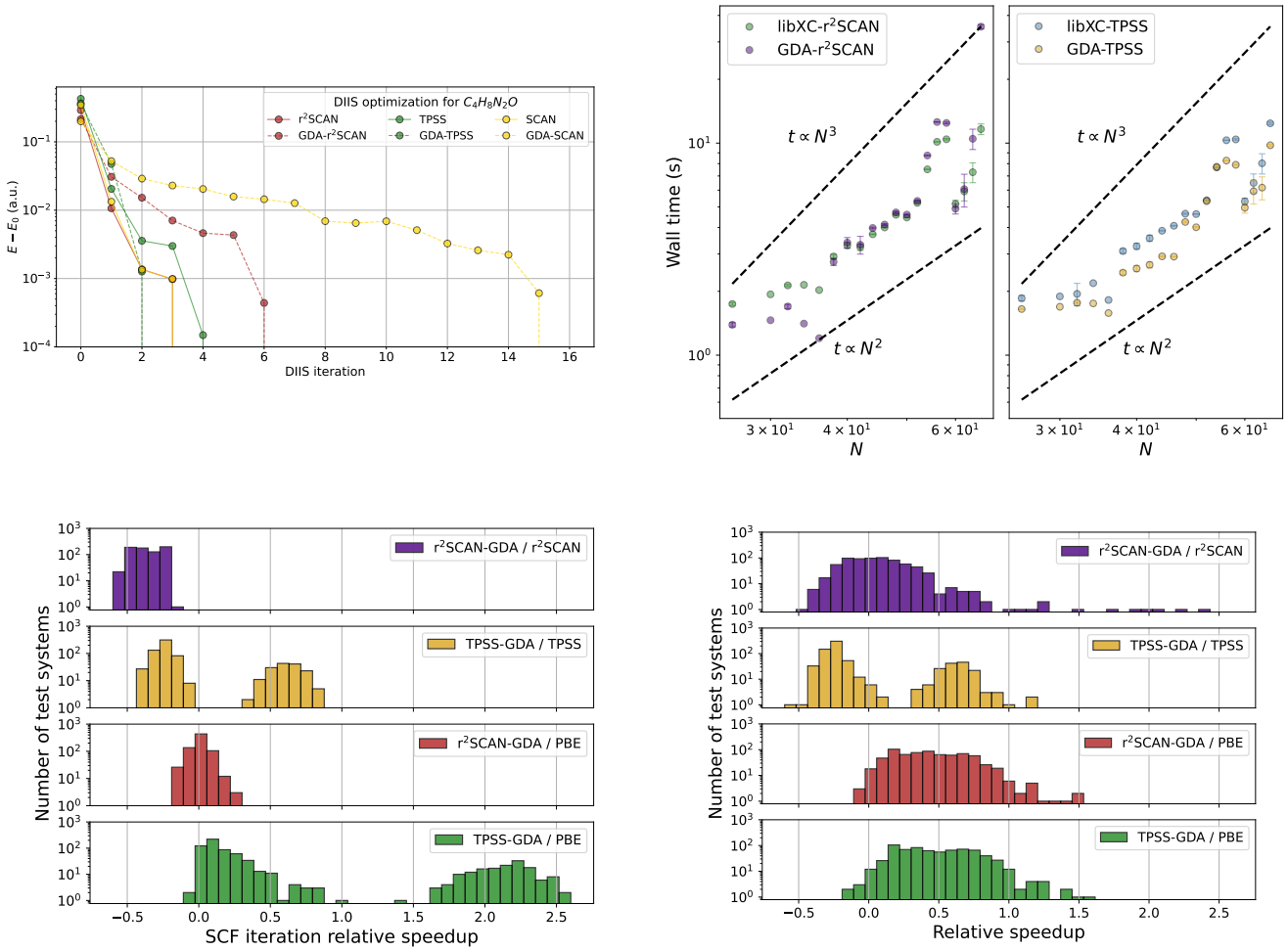

FIG. 3. Convergence and numerical performance properties of the GDA functionals. **Top left**: Convergence properties of the DIIS [19] optimization used in PySCF [17, 18], on an example of a random molecule from the test dataset. **Top right**: Total wall clock time required for SCF convergence as a function of the number of electrons N. We observe scaling consistent with the expected $\mathcal{O}(N^3)$ in the large-N limit. **Bottom left**: Relative speedups defined in Eq. 18 for individual SCF iterations. We observe that the GDA approximation offers up to 50% faster SCF iterations while still not being as efficient as the representative pure-density functional (PBE). **Bottom right**: Relative speedups for total wall clock times. We see that, within our hardware constraints, the GDA approximation offers minor speedups at best. We note that these results are extremely hardware and software dependent. Additionally, we expect that the more fine-tuned models trained on specialized data sets will improve in performance as well as benefit from more optimizations in the GDA code under development.

convergence accuracy and target precision may play an important role in determining the outcomes of such comparisons because modern GPU hardware tends to be heavily optimized for single-precision floating point operations.

Our testing setup reflects a *workstation* setup with an AMD Ryzen Threadripper 7970X 32-core CPU, Nvidia RTX 4500 (Ada Generation) GPU. Numerical run-time data was collected from independent calculations on 10% of the total QM7 [24, 25] dataset discussed in the main text. Heuristics on relative runtime speedups, data on SCF convergence, and individual iteration efficiency can be found in Fig. 3.

To quantify the performance difference, we define the *relative speedup* as

$$r = \frac{t_{\mathrm{X}} - t_{\mathrm{ref}}}{t_{\mathrm{ref}}} \tag{18}$$

for a DFT calculation taking t_X time using the GDA functional X, compared against a reference calculation t_{ref}. In Fig. 3, we compare the performance of GDA approximations against target meta-GGAs as well as PBE [26] as a representative of the pure-density functional (GGA) family.

Within the scope of our tests, we observe that GDA models are more efficient than their parent functionals on average, based on timings of individual SCF iterations. In the case of TPSS, we observe a more consistent performance

boost (up to 50%), in addition to the results presented in the main text. In terms of total (wall clock) times, we see that GDA approximations offer moderate speedups for the QM7 training set by requiring a few extra SCF iterations than reference functionals. Heuristically, we observe that this can be controlled by changing the gradient penalty term coupling λ in the cost function presented in the main text. We expect that the transferability of GDA models allows fine-tuning on more specific datasets where additional performance gains can be accessed.

Within the available hardware constraints, we observe that the overall performance gain of GDA approximation puts them somewhere between traditional meta-GGA and GGA functionals, with overall scaling still consistent with the expected asymptotic $\mathcal{O}(N^3)$.

IV. DENSITY FUNCTIONAL THEORY

A. Kohn-Sham density functional theory

As set up in the main text, we consider isolated molecular systems. A DFT calculation [27] outputs an approximation to the ground state density n_0 minimizing the total energy: $n_0(\mathbf{r}) = \mathrm{argmin}_n\, E[n]$ where

$$E[n] = T[n] + E_{\mathrm{ext}}[n] + E_H[n] + E_{\mathrm{xc}}[n] \tag{19}$$

Known terms in the total energy functional $E[n]$ given in Eq. 19 are the external contribution $E_{\mathrm{ext}}[n]$

$$E_{\mathrm{ext}}[n] = \int \mathrm{d}^3\mathbf{r}\, n(\mathbf{r})\, v_{\mathrm{ext}}(\mathbf{r}) \tag{20}$$

capturing the effects of the atomic Coulomb interaction $v_{\mathrm{ext}}(\mathbf{r})$ and the direct Hartree component capturing the classical electronic Coulomb interaction:

$$E_{\mathrm{H}}[n] = \frac{1}{2}\int \mathrm{d}^3\mathbf{r} \int \mathrm{d}^3\mathbf{r}'\, \frac{n(\mathbf{r})n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}\ . \tag{21}$$

Kohn-Sham DFT [28] framework assumes that the density n_0 comes from an effective system of non-interacting electrons with orbitals $\Psi = \{\psi_k(\mathbf{r})\,|\,k = 1,\ldots,N\}$ where N is the number of electrons in the system. In that case, the kinetic term in Eq. 19 evaluates to

$$T[n] \;\mapsto\; T_{\mathrm{KS}}[n] = \sum_a n_a \int \mathrm{d}^3\mathbf{r}\ \psi_a^*(\mathbf{r}) \left(-\frac{1}{2}\nabla^2\right) \psi_a(\mathbf{r}) = \frac{1}{2}\sum_a n_a \int \mathrm{d}^3\mathbf{r}\, |\nabla\psi_a(\mathbf{r})|^2\ . \tag{22}$$

After the desired kinetic and XC functional has been specified, the constrained minimization of the total energy functional given in Eq. 19 can proceed, enforcing orbital normalization with Lagrange multipliers ϵ_a:

$$\mathcal{L}[\Psi] = E[\Psi] + \sum_a \epsilon_a \int \mathrm{d}^3\mathbf{r}\, |\psi_a(\mathbf{r})|^2 = T_{\mathrm{KS}}[\Psi] + U[\Psi] + \sum_a \epsilon_a \int \mathrm{d}^3\mathbf{r}\, |\psi_a(\mathbf{r})|^2 \tag{23}$$

where we choose to trivially rewrite all density functionals as orbital functionals and collect all of the different kinds of potential energy into $U[\Psi] = E_{\mathrm{ext}}[\Psi] + E_{\mathrm{H}}[\Psi] + E_{\mathrm{xc}}[\Psi]$. Trivial functional differentiation yields:

$$\frac{\delta\mathcal{L}[\Psi]}{\delta\psi_a^*(\mathbf{r})} = \left(-\frac{1}{2}\nabla^2 + v_{\mathrm{eff}}[n](\mathbf{r}) - \epsilon_a\right)\psi_a(\mathbf{r}) \quad\Longrightarrow\quad \left(-\frac{1}{2}\nabla^2 + v_{\mathrm{eff}}[n](\mathbf{r})\right)\psi_a(\mathbf{r}) = \epsilon_a\,\psi_a(\mathbf{r})\,, \tag{24}$$

known as the Kohn-Sham (KS) equation. Mathematically speaking, the KS Eq. 24 is a non-linear partial differential equation for the unknown orbitals Ψ. The source of non-linearity, other than the classical Coulomb term, comes from exchange and correlation effects built into the approximate functional E_{xc} that gives rise to the effective one-electron potential

$$v_{\mathrm{eff}}(\mathbf{r}) = \frac{\delta U[n]}{\delta n(\mathbf{r})} = v_{\mathrm{ext}}(\mathbf{r}) + \int \mathrm{d}^3\mathbf{r}'\, \frac{n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} + \frac{\delta E_{\mathrm{xc}}[n]}{\delta n(\mathbf{r})}\,. \tag{25}$$

The last term in Eq. 25 is sometimes labeled as the XC potential $v_{\mathrm{xc}}(\mathbf{r})$ and in it highlights the importance of being able to calculate functional derivatives of the XC functional efficiently.

Numerically, Eq. 24 is usually solved in a self-consistent (SCF) manner, ensuring that $n(\mathbf{r}) = \sum_a n_a|\psi_a(\mathbf{r})|^2$. The SCF procedure takes the form of fixed-point iteration of the following two steps, starting from the initial guess for $n(\mathbf{r})$:

- Update orbital estimates Ψ by solving Eq. 24 for a fixed density.
- Update density estimate through $n(\mathbf{r}) = \sum_a n_a |\psi_a(\mathbf{r})|^2$.

The loop is terminated after successive density and energy estimates stop changing beyond a given tolerance.

B. Calculations in the basis of atomic orbitals

For isolated molecular systems, Gaussian basis sets are a common approach of representing atomic orbitals used in density expansions. In this subsection, we derive all of the expressions used during training to evaluate the GDA effective one-particle Hamiltonian

$$H_{\text{eff}} = \frac{\mathbf{p}^2}{2} + v_{\text{eff}}[n](\mathbf{r}) \tag{26}$$

from the neural-network functional using automatic differentiation, with $\mathbf{p}$ being the one-particle momentum operator. With the definition of Eq. 26, the KS equations simply read $H_{\text{eff}} |\psi_a\rangle = \epsilon_a |\psi_a\rangle$.

In the following, we use Greek indices $\{\mu, \nu, \ldots\}$ for the fixed atomic orbital (AO) basis and latin $\{a, b, \ldots\}$ indices for the molecular orbital (MO) basis. The two bases are related by the LCAO (linear combination of atomic orbitals) coefficients C as $|\psi_a\rangle = \sum_\mu C_{\mu a} |\chi_\mu\rangle$. The density matrix Γ is usually defined in the AO basis through

$$n(\mathbf{r}) = \sum_{\mu\nu} \Gamma_{\mu\nu} \chi_\mu(\mathbf{r}) \chi_\nu(\mathbf{r}) \, . \tag{27}$$

The full effective Hamiltonian matrix in the AO basis decomposes as:

$$F_{\mu\nu} = \langle \chi_\mu | H_{\text{eff}} | \chi_\nu \rangle = \frac{\partial E}{\partial \Gamma_{\mu\nu}} = T_{\mu\nu} + V_{\mu\nu} + J_{\mu\nu} + X_{\mu\nu} \, , \tag{28}$$

where T, V, J, and X are kinetic, nuclear, Coulomb and XC matrices, respectively. These quantities are readily available in quantum chemistry software packages and we save T as a part of the training dataset. The Kohn-Sham kinetic energy functional is of special interest in this work. Employing partial integration, we get

$$\begin{aligned}
T[n] &= \sum_a n_a \langle \psi_a | \frac{\mathbf{p}^2}{2} | \psi_a \rangle = \\
&= \sum_a n_a \int \mathrm{d}^3\mathbf{r} \, \psi_a^*(\mathbf{r}) \left(-\frac{1}{2}\nabla^2 \right) \psi_a(\mathbf{r}) = \frac{1}{2} \sum_a n_a \int \mathrm{d}^3\mathbf{r} \, |\nabla \psi_a(\mathbf{r})|^2 = \\
&= \sum_{\mu\nu} \left(\sum_a n_a C_{\mu a} C_{\nu a} \right) \times \frac{1}{2} \int \mathrm{d}^3\mathbf{r} \, \nabla\chi_\mu(\mathbf{r}) \cdot \nabla\chi_\nu(\mathbf{r}) = \sum_{\mu\nu} \Gamma_{\mu\nu} T_{\mu\nu} = \mathrm{Tr}\left(\Gamma T\right) \, ,
\end{aligned}$$

where we identify the kinetic energy matrix $T_{\mu\nu}$ that goes into the cost function defined in the main text. Other similar matrices in the AO basis work out to be:

$$\begin{gathered}
T_{\mu\nu} = \frac{1}{2} \int \mathrm{d}^3\mathbf{r} \, \nabla\chi_\mu(\mathbf{r}) \cdot \nabla\chi_\nu(\mathbf{r}) \, , \quad V_{\mu\nu} = \int \mathrm{d}^3\mathbf{r} \, v_{\text{ext}}(\mathbf{r}) \chi_\mu(\mathbf{r}) \chi_\nu(\mathbf{r}) \quad \text{and} \\
J_{\mu\nu} = \sum_{\alpha\beta} \Gamma_{\alpha\beta} \int \mathrm{d}^3\mathbf{r} \int \mathrm{d}^3\mathbf{r}' \, \frac{\chi_\mu(\mathbf{r}) \chi_\nu(\mathbf{r}) \chi_\alpha(\mathbf{r}') \chi_\beta(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \, .
\end{gathered} \tag{29}$$

The XC matrix X is the only component we need to evaluate using from the GDA functional in order to regularize the training loop. We only need to evaluate the derivative of the total XC energy with respect to the input density matrix because:

$$\hat{X}_{\mu\nu} = \frac{\partial \hat{E}_{\text{xc}}}{\partial \Gamma_{\mu\nu}} = \int \mathrm{d}^3\mathbf{r} \, \frac{\delta \hat{E}_{\text{xc}}}{\delta n(\mathbf{r})} \frac{\partial n(\mathbf{r})}{\partial \Gamma_{\mu\nu}} = \int \mathrm{d}^3\mathbf{r} \, \hat{v}_{\text{xc}}(\mathbf{r}) \, \chi_\mu(\mathbf{r}) \chi_\nu(\mathbf{r}) \, . \tag{30}$$

where we put hats on quantities estimated using the GDA functional approximation. These derivatives can be calculated efficiently using automatic differentiation at the cost of evaluating the functional itself.

---

[1] Shuhao Cao, "Choose a Transformer: Fourier or Galerkin," (2021), arXiv:2105.14995 [cs, math].

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention Is All You Need," (2017), arXiv: 1706.03762.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," (2020), arXiv: 2010.11929.

[4] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu, "RoFormer: Enhanced Transformer with Rotary Position Embedding," (2023), arXiv:2104.09864 [cs].

[5] Zijie Li, Kazem Meidani, and Amir Barati Farimani, "Transformer for Partial Differential Equations' Operator Learning," (2023), arXiv:2205.13671 [cs].

[6] Noam Shazeer, "GLU Variants Improve Transformer," (2020).

[7] Stefan Elfwing, Eiji Uchibe, and Kenji Doya, "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning," (2017), arXiv: 1702.03118.

[8] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton, "Layer Normalization," (2016), arXiv: 1607.06450.

[9] Diederik P. Kingma and Jimmy Lei Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (International Conference on Learning Representations, ICLR, 2015).

[10] 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," (2019), arXiv:1908.03265.

[11] Ilya Loshchilov and Frank Hutter, "Fixing Weight Decay Regularization in Adam," CoRR **abs/1711.05101** (2017), arXiv: 1711.05101.

[12] Ali Rahimi and Benjamin Recht, "Random Features for Large-Scale Kernel Machines," in *Advances in Neural Information Processing Systems*, Vol. 20, edited by J. Platt, D. Koller, Y. Singer, and S. Roweis (Curran Associates, Inc., 2007).

[13] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," (2020), arXiv: 2003.08934.

[14] Kyle Gao, Yina Gao, Hongjie He, Dening Lu, Linlin Xu, and Jonathan Li, "NeRF: Neural Radiance Field in 3D Vision, A Comprehensive Review," (2022), arXiv: 2210.00379.

[15] Jianqiao Zheng, Sameera Ramasinghe, and Simon Lucey, "Rethinking Positional Encoding," (2021), arXiv: 2107.02561.

[16] Yang Li, Si Si, Gang Li, Cho-Jui Hsieh, and Samy Bengio, "Learnable Fourier Features for Multi-Dimensional Spatial Positional Encoding," (2021), arXiv: 2106.02795.

[17] 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, Sebastian Wouters, and Garnet Kin-Lic Chan, "PySCF: the Python-based simulations of chemistry framework," WIREs Computational Molecular Science **8**, e1340 (2018).

[18] Qiming Sun, Xing Zhang, Samragni Banerjee, Peng Bao, Marc Barbry, Nick S. Blunt, Nikolay A. Bogdanov, George H. Booth, Jia Chen, Zhi-Hao Cui, Janus J. Eriksen, Yang Gao, Sheng Guo, Jan Hermann, Matthew R. Hermes, Kevin Koh, Peter Koval, Susi Lehtola, Zhendong Li, Junzi Liu, Narbe Mardirossian, James D. McClain, Mario Motta, Bastien Mussard, Hung Q. Pham, Artem Pulkin, Wirawan Purwanto, Paul J. Robinson, Enrico Ronca, Elvira R. Sayfutyarova, Maximilian Scheurer, Henry F. Schurkus, James E. T. Smith, Chong Sun, Shi-Ning Sun, Shiv Upadhyay, Lucas K. Wagner, Xiao Wang, Alec White, James Daniel Whitfield, Mark J. Williamson, Sebastian Wouters, Jun Yang, Jason M. Yu, Tianyu Zhu, Timothy C. Berkelbach, Sandeep Sharma, Alexander Yu. Sokolov, and Garnet Kin-Lic Chan, "Recent developments in the PySCF program package," The Journal of Chemical Physics **153**, 024109 (2020).

[19] Péter Pulay, "Convergence acceleration of iterative sequences. the case of scf iteration," Chemical Physics Letters **73**, 393–398 (1980).

[20] Miguel A.L. Marques, Micael J.T. Oliveira, and Tobias Burnus, "Libxc: A library of exchange and correlation functionals for density functional theory," Computer Physics Communications **183**, 2272–2281 (2012).

[21] Jianwei Sun, Richard C. Remsing, Yubo Zhang, Zhaoru Sun, Adrienn Ruzsinszky, Haowei Peng, Zenghui Yang, Arpita Paul, Umesh Waghmare, Xifan Wu, Michael L. Klein, and John P. Perdew, "SCAN: An Efficient Density Functional Yielding Accurate Structures and Energies of Diversely-Bonded Materials," (2015), arXiv: 1511.01089.

[22] James W. Furness, Aaron D. Kaplan, Jinliang Ning, John P. Perdew, and Jianwei Sun, "Accurate and Numerically Efficient r ^{2} SCAN Meta-Generalized Gradient Approximation," The Journal of Physical Chemistry Letters **11**, 8208–8215 (2020).

[23] Jianmin Tao, John P. Perdew, Viktor N. Staroverov, and Gustavo E. Scuseria, "Climbing the Density Functional Ladder: Nonempirical Meta–Generalized Gradient Approximation Designed for Molecules and Solids," Physical Review Letters **91**, 146401 (2003).

[24] Lorenz C. Blum and Jean-Louis Reymond, "970 Million Druglike Small Molecules for Virtual Screening in the Chemical Universe Database GDB-13," Journal of the American Chemical Society **131**, 8732–8733 (2009).

[25] Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O. Anatole Von Lilienfeld, "Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning," Physical Review Letters **108**, 058301 (2012).

[26] John P. Perdew, Kieron Burke, and Matthias Ernzerhof, "Generalized Gradient Approximation Made Simple," Physical Review Letters **77**, 3865–3868 (1996).

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

[28] W. Kohn and L. J. Sham, "Self-Consistent Equations Including Exchange and Correlation Effects," Physical Review **140**, A1133–A1138 (1965).