

# An Investigation of Physics Informed Neural Networks to solve the Poisson-Boltzmann Equation in Molecular Electrostatics

Martín A. Achondo,<sup>1</sup> Jehanzeb H. Chaudhry,<sup>2</sup> and Christopher D. Cooper<sup>1, a)</sup>

<sup>1)</sup>*Department of Mechanical Engineering, Universidad Técnica Federico Santa María, Valparaíso, Chile.*

<sup>2)</sup>*Department of Mathematics and Statistics, University of New Mexico, Albuquerque, NM, United States*

(\*Electronic mail: christopher.cooper@usm.cl.)

(\*Electronic mail: jehanzeb@unm.edu.)

(Dated: 1 October 2024)

Physics-informed neural networks (PINN) is a machine learning (ML)-based method to solve partial differential equations that has gained great popularity due to the fast development of ML libraries in the last few years. The Poisson-Boltzmann equation (PBE) is widely used to model mean-field electrostatics in molecular systems, and in this work we present a detailed investigation of the use of PINN to solve the PBE. Starting from a multidomain PINN for the PBE with an interface, we assess the importance of incorporating different features into the neural network architecture. Our findings indicate that the most accurate architecture utilizes input and output scaling layers, a random Fourier features layer, trainable activation functions, and a loss balancing algorithm. The accuracy of our implementation is of the order of  $10^{-2}$ – $10^{-3}$ , which is similar to previous work using PINN to solve other differential equations. We also explore the possibility of incorporating experimental information into the model, and discuss challenges and future work, especially regarding the nonlinear PBE. Along with this manuscript, we are providing an open-source implementation to easily perform computations from a PDB file. We hope this work will motivate application scientists into using PINN to study molecular electrostatics, as ML technology continues to evolve at a high pace.

## I. INTRODUCTION

Implicit solvation is a widely used approach in molecular modeling and considers the solvent as a continuum. This massively decreases the number of degrees of freedom of the system, and hence, the computational cost of mean-field calculations. In electrostatics, a widely used implicit method is the Poisson-Boltzmann equation (PBE)<sup>1,2</sup> solved on a multi-region domain, where the dielectric constant and salt concentration have a sharp variation along interfaces. The PBE has been solved numerically for decades using finite differences (FDM)<sup>3–7</sup>, finite elements (FEM)<sup>8–11</sup>, boundary elements (BEM)<sup>12–19</sup>, (semi) analytical<sup>20–23</sup>, and hybrid approaches<sup>24–26</sup>, proving useful for a large community.

Scientific machine learning (SciML) has seen tremendous recent interest in utilizing tools from computer science, mathematics and statistics to solve complex scientific and engineer-

ing problems. SciML encompasses a wide array of methodologies such as digital twins<sup>27</sup>, data assimilation<sup>28</sup>, Bayesian inverse analysis<sup>29</sup>, model reduction<sup>30</sup>, physics-informed machine learning<sup>31</sup> etc. In particular, a prominent and widely used physics-informed machine learning approach to solve Partial Differential Equations (PDEs) is the physics-informed neural networks (PINN).<sup>32–34</sup> PINN represents the approximate solution of the differential equation with a neural network, where the network's parameters are optimized using a loss function that contains the PDE residual. PINN has attracted a lot of interest from the computational mathematics community, and has been used in various applications, such as fluid mechanics,<sup>35</sup> heat transfer,<sup>36</sup> electromagnetism,<sup>37</sup> acoustics,<sup>38</sup> Lie-Poisson systems,<sup>39</sup> among others.

The definition of the residual gives rise to different variations of PINN, for example, by writing it in variational<sup>40,41</sup> or boundary integral<sup>42,43</sup> form. There are also special forms of PINN that are designed for domain decomposition, for example, extended PINN (XPINN),<sup>44</sup> conservative PINN (cPINN),<sup>45</sup> and distributed PINN (DPINN),<sup>46</sup> among others.

---

<sup>a)</sup>Also at Centro Científico Tecnológico de Valparaíso (CCTVal), Universidad Técnica Federico Santa María, Valparaíso, Chile.

These usually use one neural network per region, and they differ in the definition of the loss functions and the treatment of the interface conditions. These methods have been adapted to solve linear elliptic partial differential equations with interfaces<sup>47–53</sup>, where domain decomposition enhances precision by accounting for particularities of the solution in each subdomain. Additionally, there are theoretical studies that analyze their convergence<sup>54</sup> and error bounds<sup>52</sup> for elliptic interface problems. Although the aforementioned works vary in their choice of optimization algorithms, parameters, interface conditions, and neural network architectures, they all use a domain decomposition PINN strategy, and we employ a similar strategy in devising a PINN for the PBE.

The Poisson-Boltzmann model considered in this work is an example of an elliptic interface problem. Domain decomposition PINN methods (and others) have been recently applied to the PBE to compute electrostatic potentials and energies in molecular settings,<sup>49,50,55,56</sup> similar to the present work. For example, Li *et al.*<sup>55</sup> proposed a PINN approach with variational principles to solve the PBE, through a multi-scale deep neural network. In that case, there is only a single neural network, whereas our work explicitly captures the interface by using a multi-domain approach. Also, the methodology in the work by Wu and Lu<sup>49</sup> is based on using an extended multiple-gradient descent (MGD)<sup>57</sup> algorithm in a multi-domain setting. This may be a difficulty, as optimizers like MGD are not readily available in common software packages like Tensor Flow<sup>58</sup> or Torch<sup>59</sup>. In this work, we developed a PINN method for the PBE utilizing the common Adam optimizer<sup>60</sup>. Moreover, Wu and Lu define the solute-solvent interface with the van der Waals surface,<sup>61</sup> as opposed to the solvent-excluded surface (SES),<sup>62</sup> which prevails in PBE calculations, and is used in the present work. An alternative approach to solve the PBE based is based on IONet<sup>56</sup>, which is a neural network that learns the differential operator with an interface. However, this technique requires training with previously computed numerical solutions. On the other hand, Ying and co-workers<sup>50</sup> presented the multi-scale fusion network (MSFN) approach, that relies on sub-networks that approximate the solution at different frequencies (or scales), and use a least-square type loss function.

There are other physics informed machine learning techniques that have been applied to molecular electrostatics and involve the PBE, but are not designed to solve it. For example, PBML<sup>63</sup> is a neural network model that was trained with

a large dataset of PB solutions, and provides the solvation free energy from the molecular structure only. Similarly, pyPKa<sup>64</sup> uses PB calculations to feed a machine learning model that estimates pKas.

Regardless of the important progress of PINN for the PBE, further efforts are needed to understand the impact of different variants offered by PINN towards its applicability in practical cases. The present work intends to fill this gap by extensively testing different sampling strategies, architecture improvements and loss function treatments to suggest good practices of PINN for the PB equation, and identify pressing challenges moving forward. Along with this analysis, we provide a TensorFlow-based Python code called XPPBE<sup>65</sup> that computes electrostatic potentials and energies from a PDB structure using an easy interface, for interested application scientists.

We note that there are numerous traditional methods for solving the PB equation (FDM, FEM, BEM, etc.), whereas the aim of this work is to explore the recently developed ML techniques. As such, we leave a direct comparison with the traditional solvers for future work. Currently, PINN is usually slower than standard numerical schemes in most applications,<sup>66,67</sup> however, they have great potential to improve their performance, as it has happened in complex problems like weather modeling<sup>68,69</sup> where a  $10,000 \times$  speedup is observed.

In this work we focus on solving the standard PBE with PINN, however, we believe this approach can pave the way for future advancements of molecular electrostatics simulation, distinguishing itself from traditional methods. For instance, PINN can be used to solve inverse problems<sup>70</sup>, where physical properties, like permittivity, are learnable parameters. Also, PINN gives the possibility to easily incorporate experimental measurements or information from molecular dynamics simulations through a loss function, opening new doors in model development.

## II. METHODS

### A. The Poisson-Boltzmann equation

When a solute is immersed in a continuum solvent, it can be considered as a low-dielectric cavity ( $\Omega_m$ ), with relative permittivity  $\epsilon_m$  and delta-like partial charges ( $q_i$ ), inside an infinite solvent domain ( $\Omega_w$ ), as sketched in Fig. 1. The do-

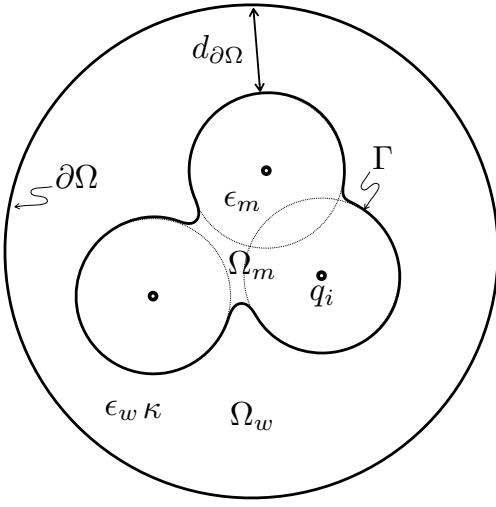


FIG. 1. Sketch of molecule for the PB model.  $\Omega_m$  is the solute domain (with partial charges  $q_i$ ),  $\Omega_w$  the solvent domain,  $\Gamma$  the interface, and  $\partial\Omega$  the edge of our domain, where we enforce boundary conditions, a minimum distance of  $d_{\partial\Omega}$  away from  $\Gamma$ .

main is truncated at  $\partial\Omega$  for practical purposes of the numerical method. In biological settings, the solvent is usually water (relative permittivity  $\epsilon_w \approx 80$ ) with salt ions that move in response to an external electric field. In equilibrium, the free ions arrange according to Boltzmann statistics, giving rise to the Poisson-Boltzmann equation. The electrostatic potential,

$$\phi(\mathbf{x}) = \begin{cases} \phi^{(m)}(\mathbf{x}), & \mathbf{x} \in \Omega_m, \\ \phi^{(w)}(\mathbf{x}), & \mathbf{x} \in \Omega_w, \end{cases} \quad (1)$$

is modeled from the following coupled system of PDEs:

$$\begin{aligned} -\epsilon_m \nabla^2 \phi^{(m)}(\mathbf{x}) &= \frac{e}{k_B T \epsilon_0} \sum_{i=1}^{n_c} q_i \delta(\mathbf{x} - \mathbf{x}_{q,i}), \quad \mathbf{x} \in \Omega_m \\ -\nabla^2 \phi^{(w)}(\mathbf{x}) + \kappa_w^2 \sinh(\phi^{(w)})(\mathbf{x}) &= 0, \quad \mathbf{x} \in \Omega_w \\ \phi^{(m)}(\mathbf{x}) &= \phi^{(w)}(\mathbf{x}), \quad \mathbf{x} \in \Gamma \\ \epsilon_m \partial_n \phi^{(m)}(\mathbf{x}) &= \epsilon_w \partial_n \phi^{(w)}(\mathbf{x}), \quad \mathbf{x} \in \Gamma \\ \phi^{(m)}(\mathbf{x} \rightarrow \infty) &= 0, \end{aligned} \quad (2)$$

where  $q_i$  is one of the  $n_c$  partial charges (represented as a Dirac  $\delta$  point charges) in the solute molecule at locations  $\mathbf{x}_{q,i}$ ,  $\kappa_w$  is the inverse of the Debye length, and the electrostatic potential  $\phi$  is nondimensionalized by  $\frac{k_B T}{e}$  (Boltzmann constant times temperature, divided by the elementary charge). The interface  $\Gamma$  is usually defined either as the solvent accessible,<sup>71</sup> solvent excluded,<sup>62</sup> van der Waals<sup>61</sup>, or Gaussian<sup>72</sup> surfaces. In this

work, we use the solvent-excluded surface (SES). The symbol  $\partial_n$  corresponds to the derivative in the direction of an outward-facing normal to  $\Gamma$ .

Eq. (2) is challenging to solve numerically because of the singularities at the charge's locations and interface conditions that are enforced in a complex geometry. There are several regularization techniques of the PB equation<sup>9,73–75</sup> that alleviate the issue of the singularities, for example, by solving for a so-called *regular* or *reaction* potential, which is a difference between the electrostatic potential and the Coulomb potential. The Coulomb potential corresponds to the electrostatic potential due to a collection of point charges in free space, which (nondimensionalized by  $\frac{k_B T}{e}$ ) is

$$g_C(\mathbf{x}) = \frac{e}{k_B T} \sum_i^{n_c} \frac{q_i}{4\pi \epsilon_m \epsilon_0 |\mathbf{x} - \mathbf{x}_{q,i}|}. \quad (3)$$

The regular potential  $\psi$  defined on the entire domain  $\Omega$  is,

$$\psi = \phi - g_C, \quad \text{in } \Omega. \quad (4)$$

In our case, we use a regularized version of the PB equation that only decomposes the electrostatic potential into singular and non-singular components in  $\Omega_m$ . That is, we set

$$\psi^{(m)} = \phi^{(m)} - g_C, \quad \text{in } \Omega_m. \quad (5)$$

Moreover, in many practical applications, such as proteins, linearizing the hyperbolic sine in the PB equation yields acceptable results, leading to the linearized Regularized Poisson-Boltzmann (RPB) Equation,

$$\begin{aligned} \nabla^2 \psi^{(m)}(\mathbf{x}) &= 0, \quad \mathbf{x} \in \Omega_m \\ -\nabla^2 \phi^{(w)}(\mathbf{x}) + \kappa_w^2 \phi^{(w)}(\mathbf{x}) &= 0, \quad \mathbf{x} \in \Omega_w \\ \psi^{(m)}(\mathbf{x}) + g_C(\mathbf{x}) &= \phi^{(w)}(\mathbf{x}), \quad \mathbf{x} \in \Gamma \\ \epsilon_m (\partial_n \psi^{(m)}(\mathbf{x}) + \partial_n g_C(\mathbf{x})) &= \epsilon_w \partial_n \phi^{(w)}(\mathbf{x}), \quad \mathbf{x} \in \Gamma \\ \phi^{(m)}(\mathbf{x} \rightarrow \infty) &= 0, \end{aligned} \quad (6)$$

The linear PB formulation is widely used, however, it may lead to large errors in highly charged systems, such as nucleic acids.

The solvation energy is a useful quantity in molecular physics,<sup>76</sup> which corresponds to the free energy required to dissolve a molecule (i.e. transfer it from a vacuum into the solvent). The solvation energy has two components: a non-

polar one, to generate an uncharged solute-shaped cavity in the solvent, and a polar one, that places the charges in said cavity. The polar component of the solvation energy is commonly computed from PBE calculations, using the following equation:

$$\Delta G_{solv} = \frac{1}{2} \sum_i^{n_c} q_i \psi(\mathbf{x}_{q,i}) \quad (7)$$

which is valid for the linearized PBE.

## B. Neural Networks and PINN

A Neural Network in the context of a PINN may be considered a function  $\mathcal{N}$ , parameterized by parameters  $\theta$  from inputs  $\mathbf{x} \in \mathbb{R}^m$  to outputs  $\mathbb{R}^n$ , where  $m$  and  $n$  denote the input and output dimensions. There are numerous neural network architectures, e.g. Multi-layer Perception (MLP), Convolutional Networks, Recurrent Networks, etc.<sup>77,78</sup> Additionally, there are numerous options for both defining the solution of the PDE using a neural network and also for defining the loss function. One simple setting of using PINN to solve a PDE is to employ a MLP with a mean-squared-error (MSE) loss function and representing the PDE solution by the output of the neural network. This simple setting often forms the basic building block for solving a PDE using PINN and we describe it next.

Given a so called activation function  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ , we define  $\sigma : \mathbb{R}^d \rightarrow \mathbb{R}^d$  by defining its  $i$ th component as  $[\sigma(\mathbf{x})]_i = \sigma(\mathbf{x}_i)$  for  $\mathbf{x} \in \mathbb{R}^d$ . That is, the activation function is defined component-wise on the input vector. Common choices of the activation function are ReLU, tanh, sigmoid, etc. Given a set of  $H + 1$  integers  $d_i, i = 0, \dots, H$ , an  $H$ -layer MLP or Feed-Forward Neural network is then defined as  $\mathcal{N}(\mathbf{x}; \theta) = \mathbf{x}^{(H)}$ , where the output of the  $i$ th layer  $\mathbf{x}^{(i)}$  is defined recursively as

$$\mathbf{x}^{(i)} = \begin{cases} \mathbf{x}, & i = 0 \\ \sigma(W^{(i)} \mathbf{x}^{(i-1)} + \mathbf{b}^{(i)}), & i = 1, \dots, H-1 \\ W^{(H)} \mathbf{x}^{(H-1)} + \mathbf{b}^{(H)}, & i = H \end{cases} \quad (8)$$

Here, the weights  $W^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$  and the biases  $\mathbf{b}^{(i)} \in \mathbb{R}^{d_i}$  form the set of trainable parameters  $\theta$  for the MLP.

Let  $\mathcal{D}(u) = f$  denote a generic PDE operator on a domain  $\Omega$ , with boundary conditions  $\mathcal{B}(u) = g$ , for appropriate functions  $u, f$  and  $g$ . Then to solve a PDE using a neural network

like the MLP is to form a loss function of the form,

$$\frac{1}{N_\Omega} \sum_{\mathbf{x}_i \in S_\Omega} |\mathcal{D}(\mathcal{N}(\mathbf{x}_i; \theta)) - f(\mathbf{x}_i)|^2 + \frac{1}{N_{\partial\Omega}} \sum_{\mathbf{x}_i \in S_{\partial\Omega}} |\mathcal{B}(\mathcal{N}(\mathbf{x}_i; \theta)) - g(\mathbf{x}_i)|^2.$$

Here the  $S_\Omega \subset \Omega$  and  $S_{\partial\Omega} \subset \partial\Omega$  are sets containing  $N_\Omega$  domain and  $N_{\partial\Omega}$  boundary collocation points respectively, whereas  $|\cdot|$  refers to the Euclidean norm. Such a loss function is often referred to as the MSE loss. All differential operators acting on  $\mathcal{N}(\mathbf{x}; \theta)$  (arising from the PDE), and the derivatives of the loss function itself (needed for its minimization) are computed via automatic differentiation.<sup>79</sup> The loss function is then minimized with respect to the parameters  $\theta$  to obtain the PINN solution  $\mathcal{N}(\mathbf{x}; \theta)$ , approximating the true solution of the PDE. The minimization is usually carried out by employing a variant of a gradient descent method e.g. SGD or Adams<sup>77</sup>. The convergence properties of PINN in this framework is studied in the work by Shin and co-workers.<sup>80</sup>

## C. A Multidomain PINN for RPB

In this section we develop a multidomain PINN for the RPB which is significantly more complex than the basic PINN method outlined in Section II B. In particular, it involves a multidomain neural network architecture, a loss function that accounts for the interface conditions of the RPB, and construction of collocation points within the solute, solvent, interface and boundary regions. Later, in Section II D, we improve on this PINN architecture with a further series of enhancements.

### 1. A multidomain neural network architecture

Several recent works have applied Physics-Informed Neural Networks (PINN) to solve elliptic partial differential equations (PDEs) with interfaces.<sup>47–53</sup> These studies commonly suggest using two independent neural networks, each approximating the solution in a specific subdomain. The interface between the subdomains is handled by adding an additional term in the loss function, ensuring consistency across the boundary. Moreover, convergence and error bounds for such approximations have been established.<sup>52,54</sup>

In this work, we follow a similar approach but with certain

modifications tailored to the regularized Poisson-Boltzmann Equation given in Eq. (6). Specifically, we design a single neural network architecture with two independent branches (throughout this paper, the term branches will also be referred to as independent neural networks or simply networks). Each branch is responsible for approximating the electrostatic potential within its respective subdomain. One branch computes the reaction potential within the solute domain ( $\mathcal{N}_m$ ), while the other estimates the total potential in the solvent domain ( $\mathcal{N}_w$ ). At the simplest level, these branches are a simple MLP, however, we outline many improvements to this basic architecture in section II D. It is important to note that these branches output different types of potentials due to the specific form of the regularized PBE used in this work:

$$\begin{aligned}\psi^{(m)}(\mathbf{x}) &\approx \psi_{\theta}^{(m)}(\mathbf{x}) := \mathcal{N}_m(\mathbf{x}; \theta_m) \\ \phi^{(m)}(\mathbf{x}) &\approx \phi_{\theta}^{(w)}(\mathbf{x}) := \mathcal{N}_w(\mathbf{x}; \theta_w)\end{aligned}\quad (9)$$

Here,  $\mathcal{N}_m(\mathbf{x}; \theta_m)$  and  $\mathcal{N}_w(\mathbf{x}; \theta_w)$  represent the outputs of the solute and solvent branches, respectively, and  $\theta_m$  and  $\theta_w$  are their corresponding trainable parameters. The trainable parameters for the neural network is thus  $\theta = \theta_m \cup \theta_w$ . This setup allows for the approximation of the reaction potential at any point within the domain as a combination of these two solutions,

$$\psi_{\theta}(\mathbf{x}) = \begin{cases} \psi_{\theta}^{(m)} = \mathcal{N}_m(\mathbf{x}; \theta_m) & \mathbf{x} \in \Omega_m \\ \psi_{\theta}^{(w)} = \mathcal{N}_w(\mathbf{x}; \theta_w) - g_C(\mathbf{x}) & \mathbf{x} \in \Omega_w \\ \psi_{\theta}^{(\Gamma)} = \frac{\psi_{\theta}^{(m)} + \psi_{\theta}^{(w)}}{2} & \mathbf{x} \in \Gamma \end{cases} \quad (10)$$

The total potential is then obtained by simply adding the Coulomb potential to the reaction potential detailed earlier:

$$\phi_{\theta}(\mathbf{x}) = \psi_{\theta}(\mathbf{x}) + g_C(\mathbf{x}) \quad (11)$$

The rationale behind using a single neural network with a unified set of parameters  $\theta = \theta_m \cup \theta_w$  (instead of two independent neural networks) is that this allows us to employ only one optimizer to minimize the loss function. We have observed that this approach leads to better convergence when solving problems of this type.

## 2. Loss function

The loss function  $L(\theta; S)$  depends on the parameters  $\theta = \theta_m \cup \theta_w$  and the set of collocation points  $S$ . The set  $S$  is divided into subsets corresponding to specific regions:  $S = S_{\Omega_m} \cup S_{\Omega_w} \cup S_{\Gamma} \cup S_{\partial\Omega}$ . Here,  $S_{\Omega_m}$  and  $S_{\Omega_w}$  represent the collocation points in the solute and solvent domains, respectively, while  $S_{\Gamma}$  refers to the points at the interface, and  $S_{\partial\Omega}$  corresponds to the points on the boundary of the solvent domain (see Fig. 1). In  $S_{\partial\Omega}$ , we need to approximate that the potential decays to zero at infinity ( $\phi^{(m)}(\mathbf{x} \rightarrow \infty) = 0$  in Eq. (6)), which we do by enforcing the following Yukawa potential at those points:

$$g_Y(\mathbf{x}) = \frac{e}{k_B T} \sum_i \frac{q_i e^{-\kappa |\mathbf{x} - \mathbf{x}_{q,i}|}}{4\pi \epsilon_w \epsilon_0 |\mathbf{x} - \mathbf{x}_{q,i}|}. \quad (12)$$

Let  $N_j$  denote the count of each subset  $S_j$ . The loss function  $L(\theta; S)$  is decomposed into,

$$\begin{aligned}L(\theta; S) &= w_{\Omega_m} L_{\Omega_m}(\theta_m; S_{\Omega_m}) + w_{\Omega_w} L_{\Omega_w}(\theta_w; S_{\Omega_w}) \\ &\quad + w_{\partial\Omega} L_{\partial\Omega}(\theta_w; S_{\partial\Omega}) + w_{\Gamma_D} L_{\Gamma_D}(\theta_m, \theta_w; S_{\Gamma}) \\ &\quad + w_{\Gamma_N} L_{\Gamma_N}(\theta_m, \theta_w; S_{\Gamma})\end{aligned}\quad (13)$$

where

$$\begin{aligned}L_{\Omega_m}(\theta_m; S_{\Omega_m}) &= \frac{1}{N_{\Omega_m}} \sum_{\mathbf{x}_i \in S_{\Omega_m}} \left[ \nabla^2 \psi_{\theta}^{(m)}(\mathbf{x}_i) \right]^2, \\ L_{\Omega_w}(\theta_w; S_{\Omega_w}) &= \frac{1}{N_{\Omega_w}} \sum_{\mathbf{x}_i \in S_{\Omega_w}} \left[ \nabla^2 \phi_{\theta}^{(w)}(\mathbf{x}_i) - \kappa^2 \phi_{\theta}^{(w)}(\mathbf{x}_i) \right]^2, \\ L_{\partial\Omega}(\theta_w; S_{\partial\Omega}) &= \frac{1}{N_{\partial\Omega}} \sum_{\mathbf{x}_i \in S_{\partial\Omega}} \left[ \phi_{\theta}^{(w)}(\mathbf{x}_i) - g_Y(\mathbf{x}_i) \right]^2, \\ L_{\Gamma_D}(\theta_m, \theta_w; S_{\Gamma}) &= \frac{1}{N_{\Gamma}} \sum_{\mathbf{x}_i \in S_{\Gamma}} \left[ \phi_{\theta}^{(w)}(\mathbf{x}_i) - \psi_{\theta}^{(m)}(\mathbf{x}_i) - g_C(\mathbf{x}_i) \right]^2, \\ L_{\Gamma_N}(\theta_m, \theta_w; S_{\Gamma}) &= \frac{1}{N_{\Gamma}} \sum_{\mathbf{x}_i \in S_{\Gamma}} \left[ \epsilon_w \frac{\partial}{\partial n} \left( \phi_{\theta}^{(w)}(\mathbf{x}_i) \right) \right. \\ &\quad \left. - \epsilon_m \frac{\partial}{\partial n} \left( \psi_{\theta}^{(m)}(\mathbf{x}_i) + g_C(\mathbf{x}_i) \right) \right]^2.\end{aligned}$$

Each term in the loss function corresponds to a specific region of the domain:  $\Omega_m$  (solute domain),  $\Omega_w$  (solvent domain),  $\partial\Omega$  (boundary of the solvent domain), and  $\Gamma$  (interface between the domains). Each loss term is weighted by a factor  $w_j$ , where the index  $j$  varies over  $\{\Omega_m, \Omega_w, \partial\Omega, \Gamma_D, \Gamma_N\}$ , to balance its contribution, ensuring that all components influence

the optimization of the parameter set  $\theta$ . Notably, the term  $L_{\Omega_m}$  depends only on the parameters of the branch associated with the solute domain, while  $L_{\Omega_w}$  and  $L_{\partial\Omega}$  depend on the solvent domain. The two interface terms  $L_{\Gamma_D}$  and  $L_{\Gamma_N}$ , which enforce continuity of the potential and the electric displacement respectively, incorporate contributions from both neural networks.

In addition to the primary loss terms, additional terms can be introduced to incorporate known approximations, experimental results, or other relevant information. For example, a loss term based on known approximation can be defined as:

$$L_{data}(\theta; S_{data}) = \frac{1}{N_{data}} \sum_{\mathbf{x}_i \in S_{data}} [\phi_\theta(\mathbf{x}_i) - \phi^\dagger(\mathbf{x}_i)]^2 \quad (14)$$

This term compares the predicted potential  $\phi_\theta(\mathbf{x}_i)$  with known approximations  $\phi^\dagger(\mathbf{x}_i)$  at the collocation points  $S_{data}$ .

### 3. Construction of collocation points

The molecular surface or interface  $\Gamma$ , modeled by a solvent-excluded surface (SES), often has a complex geometry and hence it is not straightforward to construct the set of collocation points  $S = S_{\Omega_m} \cup S_{\Omega_w} \cup S_\Gamma \cup S_{\partial\Omega}$ . To accomplish this we first create a surface triangular mesh of the SES, and then create a volumetric tetrahedral mesh of the domain  $\Omega$  which conforms to the SES (that is, the only intersection between a tetrahedron and the SES is a triangular face of the tetrahedron). Then we consider four sub-meshes: two tetrahedral sub-meshes corresponding to  $\Omega_m$  and  $\Omega_w$ , and two triangular sub-meshes corresponding to  $\Gamma$  and  $\partial\Omega$  (see Fig. 2). Then we select elements from each sub-mesh, and sample a point per each selected element randomly to generate  $S_{\Omega_m}, S_{\Omega_w}, S_\Gamma$  and  $S_{\partial\Omega}$ , as sketched by Fig. 4. An example of the resulting set of collocation nodes for arginine is presented by Fig. 3.

This approach allows us to construct random samples without parameterizing the domains, while only requiring the discretization of the molecular surface (SES) which is a well-established process. The distribution of collocation points throughout the domains depends on the distribution of the mesh elements, giving us full control over point density in different regions. This allows us to concentrate more points in areas where residuals are typically higher, such as at the interface. Additionally, the process may be repeated for selected elements to increase the density of collocation points in spe-

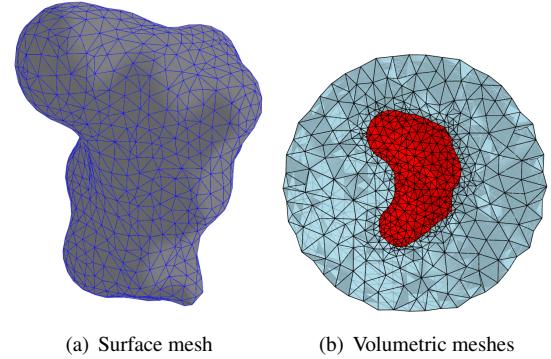


FIG. 2. Examples of (a) surface mesh (b) volumetric meshes for arginine.

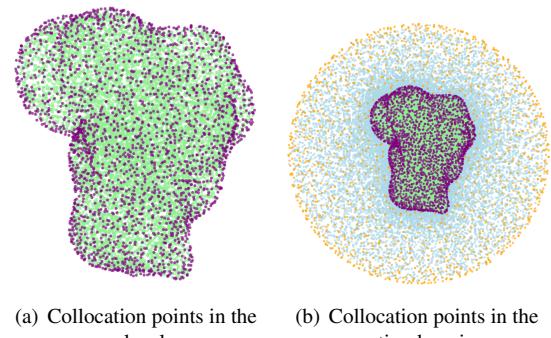


FIG. 3. Examples of the collocation points obtained using the meshes showed in Fig 2. Green nodes correspond to  $S_{\Omega_m}$ , purple nodes are  $S_\Gamma$ , light blue nodes are  $S_{\Omega_w}$ , and orange nodes are  $S_{\partial\Omega}$ .

cific regions. Finally, the density of elements in the mesh is chosen such that subdomains with larger size have more elements. This results in an increase of collocation points as the size of the subdomain (solute, solvent, molecular surface and boundary) is increased, similar to the method described by Jiang and co-workers.<sup>52</sup>

The approach to construct collocation points is consistent with current research in PINN and can be adapted to include importance sampling techniques<sup>81</sup> and residual-based adaptive sampling methods<sup>82</sup> by adjusting the probability distribution for sampling within each mesh element.

### D. An enhanced PINN architecture for RPB

We discuss improvements to the PINN architecture in this section. These enhancements lead to significant increases in the accuracy and efficiency of the computed solution, as we demonstrate in Section III. The architecture of our PINN algorithm for solving the PB equation is illustrated in Figure

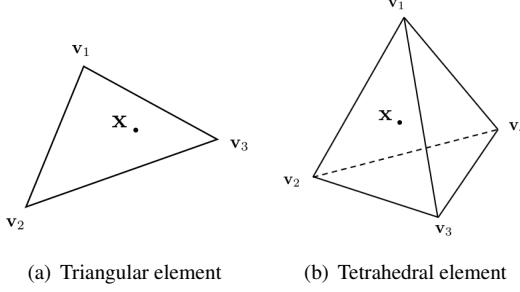


FIG. 4. Schematic of a random point  $X$  inside a (a) triangular element (b) tetrahedral element.

5. The input is passed to either  $\mathcal{N}_m$  or  $\mathcal{N}_w$  depending on its location. The input then successively passes through an input scaling layer (Section II D 1), a Fourier feature layer (Section II D 2), hidden layers with trainable activation function (II D 3) and finally an output scaling layer (Section II D 1). The output of the network, along with derivatives computed using automatic differentiation, is then used to compute the loss function. The minimization of the loss function employs a loss balancing algorithm (Section II D 4). The parameters of the network are updated and the process repeated.

### 1. Scaling layers

It is well known that scaling helps convergence of the neural network training.<sup>83</sup> Two scaling layers each for the networks  $\mathcal{N}_m$  and  $\mathcal{N}_w$  are employed to normalize the inputs and outputs, improving its convergence during training. This ensures that the values entering and leaving the network are between -1 and 1, or at least close to this range.

The input scaling is performed before the hidden layers (or the Fourier features layer if that is used). Given an input  $\mathbf{x} = [x_1, x_2, x_3]$ , the input scaling is for each component  $x_i$  is,

$$x_{scaled,i} = 2 \frac{(x_i - x_{min,i})}{(x_{max,i} - x_{min,i})} - 1, \quad i \in \{1, 2, 3\}, \quad (15)$$

where  $\mathbf{x}_{min} = [x_{min,1}, x_{min,2}, x_{min,3}]$  and  $\mathbf{x}_{max} = [x_{max,1}, x_{max,2}, x_{max,3}]$  correspond to the minimum and maximum coordinates of the collocation points in the appropriate subdomain, and form hyperparameters (i.e., nontrainable parameters) for each network.

The output scaling is performed after the hidden layers. If the output of the last hidden layer is  $y_{scaled}$  (which is a scalar

for both  $\mathcal{N}_m$  and  $\mathcal{N}_w$ ), then the output scaling transform is,

$$y = \frac{y_{scaled} + 1}{2} (y_{max} - y_{min}) + y_{min}, \quad (16)$$

where the values  $y_{min}$  and  $y_{max}$  correspond to hyperparameters (i.e., nontrainable parameters) for each network and must be estimated based on approximations of the real solution in each domain. If these values,  $y_{min}$  and  $y_{max}$ , correspond to the actual maximum and minimum of the solution, then the output scaling tends to scale the values of  $y_{scaled}$  between -1 and 1. However, estimating  $y_{min}$  and  $y_{max}$  is not straightforward since the solution to the problem is not known a priori. To estimate the values of  $y_{min}$  and  $y_{max}$ , an approximation of the potential is constructed by superimposing the known solution of the Born ion,<sup>84</sup> assuming that each charge in the molecule is an independent Born ion. More precisely, let the BI function be defined as:

$$BI(q_i, R_i) = \frac{q_i}{4\pi} \left( \frac{1}{\epsilon_w(1 + \kappa_w R_i) R_i} - \frac{1}{\epsilon_m R_i} \right) \quad (17)$$

where  $R_i$  corresponds to the radius of the  $i$ th charge, the minimum and maximum reaction potentials (which are then used to obtain  $y_{min}$  and  $y_{max}$ ) are estimated with the following equations:

$$\begin{aligned} \psi_{max} &= \max_i \left( 0, BI(q_i, R_i), BI(q_i, R_i) + \sum_{j \neq i} BI(q_j, R'_{ji}) \right) \\ \psi_{min} &= \min_i \left( 0, BI(q_i, R_i), BI(q_i, R_i) + \sum_{j \neq i} BI(q_j, R'_{ji}) \right) \end{aligned}$$

where  $R'_{ji} = \|\mathbf{x}_{q,i} - \mathbf{x}_{q,j}\|$ . Note that  $(y_{min}, y_{max})$  correspond to reaction potential  $(\psi_{min}, \psi_{max})$  in  $\mathcal{N}_m$ , while the Coulomb potential  $g_C$  needs to be added in the  $\mathcal{N}_w$  (see (11)). This approximation does not guarantee that the output scaling layer will scale the solution strictly between -1 and 1, but in practice the values are close enough.

### 2. Random Fourier features layer

MLPs often suffer from a phenomenon called spectral bias, which biases the solution towards low-frequency functions, preventing the network from learning higher-frequency functions necessary to target the desired solution. This phenomenon can be mitigated using a Random Fourier Feature layer,<sup>85</sup> which maps the input signals to a high-frequency

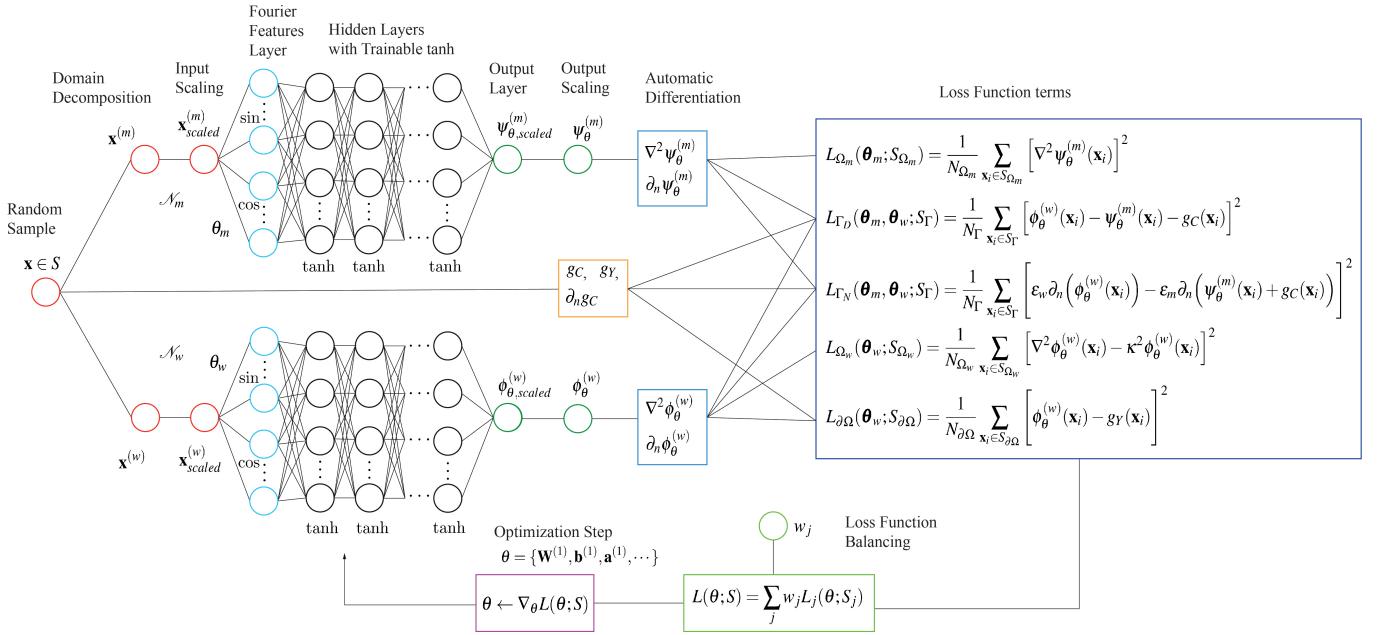


FIG. 5. Schematic of the architecture of our PINN algorithm for solving the PB equation. This method involves segregating the collocation points across the two domains, which are then fed into each branch of the neural network. Note that each branch has its own architecture, Fourier features and scaling layers. Each output contributes to the construction of the loss function  $L$ , which is minimized to find the optimal trainable parameters  $\theta$  of the neural network.

space before passing them through the neural network. Given an input  $\mathbf{x}$ , the layer is defined as follows:

$$\bar{\mathbf{x}} = \begin{bmatrix} \cos(\mathbf{B}\mathbf{x}) \\ \sin(\mathbf{B}\mathbf{x}) \end{bmatrix} \quad (18)$$

where  $\mathbf{B}$  is a matrix of shape  $m \times d$ , with  $m$  being the number of Fourier features and  $d$  the input dimension. The matrix  $\mathbf{B}$  is generated from a normal distribution  $\mathbf{B} \sim \text{Normal}(0, \sigma^2)$  and is non-trainable. This simple layer improves the PINN method's ability to learn sharp gradients and complex solutions.<sup>83</sup> In this work, we used  $m = 128$  Fourier features,  $d = 3$  corresponding to points in three dimensions, and a standard deviation of  $\sigma = 1$ .

### 3. Trainable activation function

Motivated by the work by Jagtap and co-workers,<sup>86</sup> we implemented trainable activation functions. In this approach, the activation functions in each perceptron of the neural network are associated with a trainable parameter. Specifically, in this work, we use the hyperbolic tangent function, where the train-

able parameter modifies the activation function as follows:

$$\sigma(\mathbf{x}) = \tanh(\mathbf{a} \odot \mathbf{x}), \quad (19)$$

where  $\odot$  indicates element-wise multiplication. Here,  $\mathbf{a}$  is a vector of trainable parameters (initialized to 1), which is then incorporated into the overall set of trainable parameters  $\theta$ . With this modification, the operation in the  $i$ th hidden layer is:

$$\mathbf{x}^{(i)} = \tanh\left(\mathbf{a}^{(i)} \odot (W^{(i)}\mathbf{x}^{(i-1)} + \mathbf{b}^{(i)})\right), \quad (20)$$

and the full set of parameters is defined as:

$$\theta = \{W^{(1)}, b^{(1)}, a^{(1)}, \dots, W^{(H)}, b^{(H)}, a^{(H)}\} \quad (21)$$

### 4. Loss balancing algorithm

To ensure that each term of the loss function contributes effectively to the modification of the trainable parameters  $\theta$ , we implemented a weight-adapting algorithm.<sup>83</sup> Consider the loss function as a linear combination of several loss terms:

$$L(\theta; S) = \sum_j w_j L_j(\theta; S_j) \quad (22)$$

Note that in our case the function  $L(\theta; S)$  is given in Eq. (13) and corresponds to the form given above in Eq. (22). The objective is to determine the weights  $w_j$  for each loss term  $L_j$  such that the gradient  $w_j \|\nabla_\theta L_j\|$  remains constant across all terms  $j$ :

$$C = w_j \|\nabla_\theta L_j\| \quad \forall j \quad (23)$$

To achieve this, we compute an estimator  $\hat{w}_j$  for each term:

$$\hat{w}_j = \frac{\sum_i \|\nabla_\theta L_i\|}{\|\nabla_\theta L_j\|} \quad (24)$$

Next, we update the old weight using a soft adjustment, where  $\alpha$  is a hyperparameter controlling the update rate:

$$w_{j,\text{new}} = \alpha w_{j,\text{old}} + (1 - \alpha) \hat{w}_j \quad (25)$$

In this work, we set  $\alpha = 0.7$  to balance the influence of old and new weights, applying this adjustment every  $r = 1000$  iterations

### 5. Complete Algorithm

The complete algorithm used for the PINN solution of the PBE is presented in Algorithm 1.

---

#### Algorithm 1 Algorithm for solving the PBE using PINN

---

**Input:** Molecular information and hyperparameters

```

Generate mesh from the molecular information           ▷ §II C 3
Create neural network  $\mathcal{N} = \mathcal{N}_m \cup \mathcal{N}_w$            ▷ §II C and §II D
Initialize trainable parameters  $\theta^{[0]}$                   ▷ §III
for  $k = 1$  to  $k = n$  do                                ▷ Training loop
    for  $i \in \{\Omega_m, \Omega_w, \partial\Omega, \Gamma\}$  do
        Construct subdomain collocation points  $S_i$       ▷ §II C 3
    end for
    Set  $S = S_{\Omega_m} \cup S_{\Omega_w} \cup S_{\partial\Omega} \cup S_\Gamma$  ▷ Collocation points
    Compute  $\psi_\theta, \phi_\theta$  from  $\mathcal{N}(S; \theta^{[k]})$           ▷ Eqs. (10) and (11)
    Compute loss function  $L(\theta^{[k]}; S)$                 ▷ Eqs. (13) and (22)
    Update trainable parameters  $\theta^{[k+1]}$                  ▷ Optimization step
    if mod( $k, r$ ) == 0 then
        for  $j \in \{\Omega_m, \Omega_w, \partial\Omega, \Gamma_D, \Gamma_N\}$  do
            Update weight  $w_j$  of loss term  $L_j$            ▷ §II D 4
        end for
    end if
end for

```

---

**Output:** Optimized parameters  $\theta^{[n]} (= \theta = \theta_m \cup \theta_w)$

---

We implemented Algorithm 1 in a software package and named it XPPBE. This solver requires two inputs: a .yaml file, which contains all the solver parameters (such as architectures, mesh parameters, and optimization methods, among others), and a .pdb file which contains the molecular information. A more detailed explanation of these files can be found in the tutorials available in the GitHub repository.<sup>65</sup>

### III. RESULTS AND DISCUSSION

We carry out a sequence of experiments to validate the PINN algorithm for the RPB outlined in sections II C and II D. The results are organized to demonstrate the impact of each algorithm feature on the solution, therefore providing evidence of the importance of including all of them. The analysis starts with the simple Born ion, and scales up to more complicated structures, where the influence of each component is clearer. The physical parameters were set to a solvent permittivity of  $\epsilon_w = 80$ , an inverse of the Debye length of  $\kappa = 0.125 \text{ \AA}^{-1}$ , and a solute dielectric constant of  $\epsilon_m = 2$ , except the spherical cases, where  $\epsilon_m = 1$ .

We explore a variety of PINN architectures to gauge the efficacy of different features. The base case, termed *Minimal*, is the architecture presented in Section II C and considers a fully connected multi-layer perceptron (MLP) with 4 hidden layers and 200 neurons per layer, with an hyperbolic tangent activation function. The enhancements, presented in Section II D, considered are adding the weight adjusting algorithm (WA) of section II D 4, using trainable activation functions (TF) of section II D 3, layers including Fourier features (FF) of section II D 2, and scaling of the input (SI) and output (SO) of the neural network of section II D 1. A combination of these features is indicated by the ‘+’ sign e.g., WA + SO indicates the usage of weight adjusting algorithm and output scaling. All trainable parameters were initialized using the Glorot normal distribution. We employed the Adam optimization algorithm with an exponentially decaying learning rate starting from 0.001.

The collocation points are created from surface and volume meshes (triangles and tetrahedrons, respectively) as described in section II C 3. As the molecular surface definition, we used the solvent-excluded surface (SES),<sup>62</sup> which we meshed with msms<sup>87</sup> for the spheres and arginine, and with Nanoshaper<sup>88</sup> for the bigger molecules. From those surface

definitions, we generated volumetric tetrahedral meshes with TetGen<sup>89</sup> through PyGAMer<sup>90</sup>. The external spherical surface ( $\partial\Omega$ ) was placed at a minimum distance  $d_{\partial\Omega} = 3.5 \text{ \AA}$  from  $\Gamma$ , except the full proteins (1pgb and 1uqb in Section III D, where  $d_{\partial\Omega} = 4 \text{ \AA}$ ).

For comparisons, we solved the PBE using either analytical expressions, available for spherical inclusions,<sup>84,91</sup> or BEM, through the software PBJ,<sup>19,92</sup> to compute the difference in  $\Delta G_{solv}$  by:

$$D_{G_{solv}} := \frac{|\Delta G_{solv,\theta} - \Delta G_{solv}^{ref}|}{|\Delta G_{solv}^{ref}|} \quad (26)$$

and the difference in reaction potential ( $\psi$ ):

$$D_\psi = \sqrt{\frac{\sum_i^{N_v} (\psi_\theta^{(\Gamma)}(\mathbf{x}_{v,i}) - \psi^{ref}(\mathbf{x}_{v,i}))^2}{\sum_i^{N_v} (\psi^{ref}(\mathbf{x}_{v,i}))^2}} \quad (27)$$

evaluated at the  $N_v$  vertices of the reference surface mesh ( $\mathbf{x}_{v,i}$ ). Here,

$$\Delta G_{solv,\theta} = \frac{1}{2} \sum_i^{n_c} q_i \psi_\theta^{(m)}(\mathbf{x}_{q,i}) \quad (28)$$

is the solvation energy computed with the PINN solution, and the superscript *ref* corresponds to a reference solution with either an analytical approach or BEM.

We also report training and validation losses, which correspond to the evaluation of the loss function in Eq. (13) by setting all weights to one ( $w_j = 1$ ). The training loss is computed on the collocation nodes used during training, while the validation loss is evaluated on separate points not included in the training process.

The results detailed in this section were obtained using the XPPBE software,<sup>65</sup> an open-source TensorFlow-based Python code that can easily run PBE calculations from a PDB file. All runs were performed on a workstation with two Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz with 12 cores each and 96 GB RAM memory, and a CUDA-enabled NVIDIA K40 GPU.

#### A. Born Ion

We consider the electrostatic potential of a 1 Å radius Born ion with a centered  $1e^-$  charge, using 8 different PINN archi-

tectures. Starting from the base setup (*Minimal*), we systematically add other features to evaluate their effect on accuracy. In this test, the number of collocation nodes was 1,128 in  $\partial\Omega$ , 916 in  $\Gamma$ , 8,798 in  $\Omega_w$ , and 2,353 in the  $\Omega_m$ , and the training was carried out for 20,000 iterations.

Fig. 6 compares the reaction potential computed with PINN ( $\psi_\theta$ ) with an analytical solution<sup>91</sup>. The *Minimal* setup is clearly the least accurate (Fig. 6(a)), but the results are considerably improved by including the (WA) feature (Fig. 6(b)). Including other features induce some errors in  $\Omega_m$ , however the two last architectures demonstrate better accuracy (Figs. 6(g) and 6(h)).

These results are further supported by the evolution of  $\Delta G_{solv,\theta}$ ,  $D_{G_{solv}}$ , training and validation loss, and  $D_\psi$  with iterations in Fig. 7. The figure consistently shows that the architectures (WA + TF + SI + SO) and (WA + TF + FF + SI + SO) have the lowest and smoother metrics (errors and losses) as demonstrated by the yellow and purple lines. Interestingly, the architecture with only the (WA) feature, which seemed to show excellent accuracy in Fig. 6(b), has a poor and noisy convergence in all metrics of Fig. 7. The values for the last iteration is detailed in Table I, where both  $D_{G_{solv}}$  and  $D_\psi$  arrive at a value of  $\sim 10^{-2}$ — $10^{-3}$ , even though the losses may be much smaller. These orders of magnitude for the error are consistent with previous work using PINN to solve partial differential equations.<sup>83</sup>

From this analysis, we conclude that the best two architectures include the following features: (WA + TF + SI + SO) and (WA + TF + FF + SI + SO), and we further analyze them in more complicated settings.

TABLE I. Results for the Born ion after 20,000 iterations with 8 alternative architectures. Analytical solvation energy was -164.19 kcal/mol.

Case	$D_{G_{solv}}$	$D_\psi$	Training loss	Validation loss
Minimal	5.72E-02	5.98E-02	7.10E-06	9.10E-06
WA	3.67E-03	6.18E-03	6.04E-03	5.86E-03
WA+TF	5.91E-02	6.15E-02	7.93E-06	9.78E-06
WA+FF	1.32E-02	1.22E-02	6.86E-03	7.45E-03
WA+TF+FF	1.81E-02	1.22E-02	4.21E-03	3.76E-03
WA+TF+FF+SI	1.13E-02	8.40E-03	8.27E-04	2.12E-03
WA+TF+SI+SO	5.38E-03	3.10E-03	2.44E-08	2.71E-07
WA+TF+FF+SI+SO	6.04E-03	3.27E-03	8.19E-07	2.97E-06

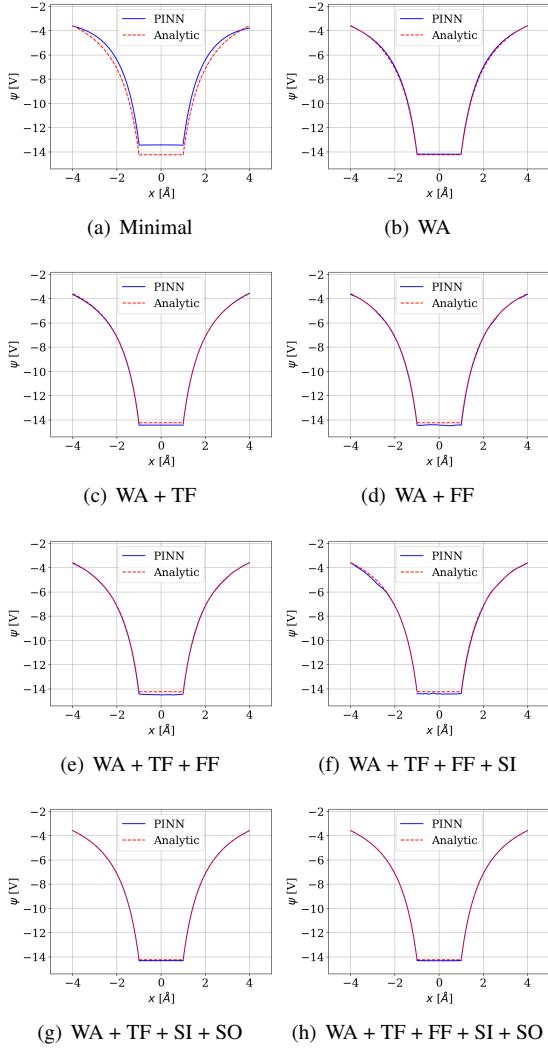


FIG. 6. Reaction potential ( $\psi$ ) along the  $x$  axis for the Born ion. The blue solid line is the PINN approximation, whereas the analytical solution is the red segmented line.

## B. Spherical molecule with off-centered charge

The only difference between the best-performing architectures for the Born ion is the incorporation of the Fourier features (FF). Here, we further analyze the impact of FF using a slightly more challenging test: a 1 Å spherical inclusion with an off-centered  $+1e^-$  charge placed 0.45 Å away from the center. This case also has an analytical solution for comparison.<sup>91</sup>

The results are shown in Figs. 8 and 9, and Table II. Even though the results in Fig. 8 and Table II are not conclusive regarding the impact of including the Fourier features, Fig. 9 gives more information. The evolution of  $\Delta G_{solv,\theta}$ ,  $D_{G_{solv}}$ , and the losses is less noisy when considering the Fourier features

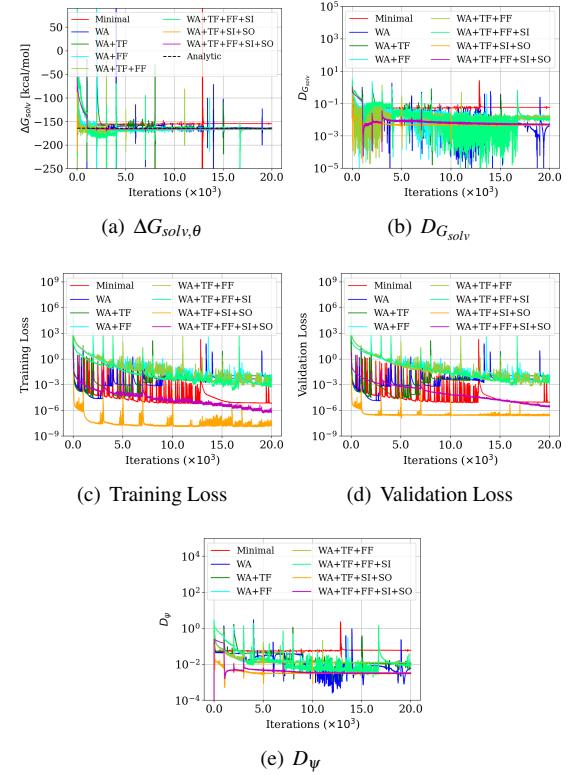


FIG. 7. Evolution of  $\Delta G_{solv,\theta}$ ,  $D_{G_{solv}}$ , training and validation loss, and  $D_\psi$  with iterations for the Born ion with 8 different architectures.

(purple line), and hence, we decided to continue our study using the architecture with all features (WA + FF + TF + SI + SO) included.

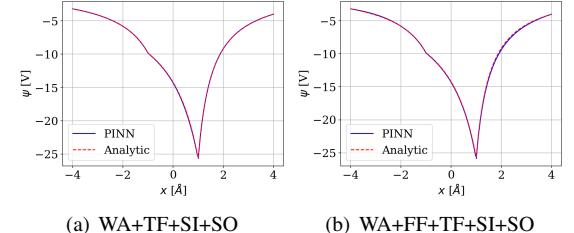


FIG. 8. Reaction potential ( $\psi$ ) on the  $x$  axis for a sphere with an off-centered charge. PINN solution in solid blue, analytical solution in red segmented.

## C. Arginine

### 1. Density of collocation nodes

Moving to a more realistic setting, we now use PINN on a single arginine<sup>93</sup> (27 atoms), and study how the density of

TABLE II. Results for the sphere with an off-centered charge after 20,000 iterations. The analytical solution was -205.55 kcal/mol.

Case	$D_{G_{solv}}$	$D_\psi$	Training loss	Validation loss
WA+TF+SI+SO	8.44E-03	3.92E-03	1.51E-05	1.38E-05
WA+TF+FF+SI+SO	5.54E-03	6.57E-03	3.53E-05	7.33E-05

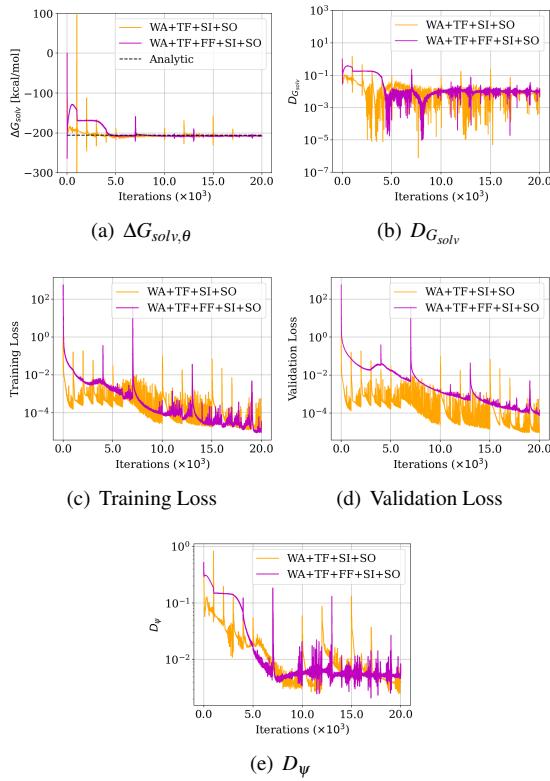


FIG. 9. Evolution of  $\Delta G_{solv,\theta}$ ,  $D_{G_{solv}}$ , training and validation loss, and  $D_\psi$  with iterations for the sphere with an off-centered charge with 2 different architectures.

collocation nodes at the surface affects the quality of the solution. Table III describes the number of collocation nodes in 4 cases (*Coarse*, *Medium*, *Fine*, and *Finest*). The collocation node distributions were obtained from surface triangulations with 0.5, 1.0, 2.0, and 4.0 vertices per  $\text{\AA}^2$  on  $\Gamma$ , respectively, 0.59 vertices per  $\text{\AA}^2$  on  $\partial\Omega$ , and tetrahedrons that conform to said triangulations, which have a maximum volume limit of 0.05  $\text{\AA}^3$  for  $\Omega_m$  and 0.6  $\text{\AA}^3$  for  $\Omega_s$ .

TABLE III. Number of collocation nodes in density study of arginine

Density	$\Omega_m$	$\Omega_w$	$\Gamma$	$\partial\Omega$
Coarse	3383	10413	282	1238
Medium	4418	10440	372	1238
Fine	6017	11596	624	1238
Finest	7568	15089	1318	1238

Table IV shows the results for the different collocation point densities. Similar to standard numerical methods, all indicators improve as the number of collocation nodes increases. Note that  $D_{G_{solv}}$  is computed against a BEM solution that uses the equivalent surface mesh for each case (*i.e.* the reference solution and definition of  $\Gamma$  is different in each case). Regardless,  $D_{G_{solv}}$  and  $D_\psi$  decrease with the number of collocation nodes, indicating that it is converging to the numerical solution computed with BEM. The latter statement is more evident in Fig. 10, where the black line converges to the red line, which corresponds to the BEM solution computed on the mesh that was used to generate the collocation points. As a reference, the blue line in Fig. 10 is a BEM solution with a surface mesh that is 4 times finer than the *Finest* case.

Fig. 11 shows the reaction potential in the  $y$  axis. From these results, it is evident that the coarsest PINN calculation (Fig 10(a)) struggles to adapt to the BEM solution, specially near the interface, however, this improves substantially for the *Medium* density in Fig. 10(b). For the two finest cases in Figs. 10(c) and 10(d), the reaction potential seems to have already converged to the BEM solution

Figs. 12 and 13 show the reaction potential on the molecular surface ( $\Gamma$ ). Similar to Fig. 11, there are no notable differences in the reaction potential of Fig. 12 beyond the *Medium* density. However, the absolute difference plots of Fig. 13 do show differences for the finer cases, which clearly present more purple regions than the *Medium* density. This indicates closer agreement between PINN and BEM as the node density increases, something that is also evidenced by Table IV.

TABLE IV. Results for collocation node density study of arginine after 20,000 iterations.

Case	$\Delta G_{solv,\theta}$ kcal/mol	$D_{G_{solv}}$	$D_\psi$	Training loss	Validation loss
Coarse	-185.1	2.34E-01	2.20E-02	1.58E-04	1.22E-04
Medium	-147.7	4.31E-02	1.41E-02	8.61E-05	9.90E-05
Fine	-146.3	5.68E-02	1.01E-02	2.73E-05	1.32E-04
Finest	-135.7	1.42E-02	8.21E-03	3.72E-06	9.84E-06

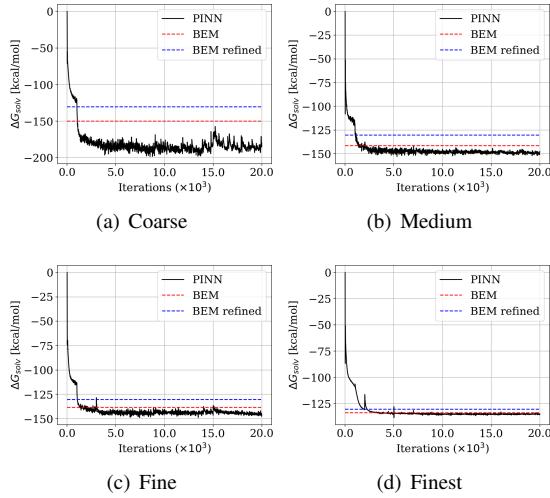


FIG. 10. Solvation energy history with iterations (black line). The red line is a BEM solution computed on the surface mesh that was used to generate the collocation nodes. The blue line corresponds to a fine-mesh BEM solution.

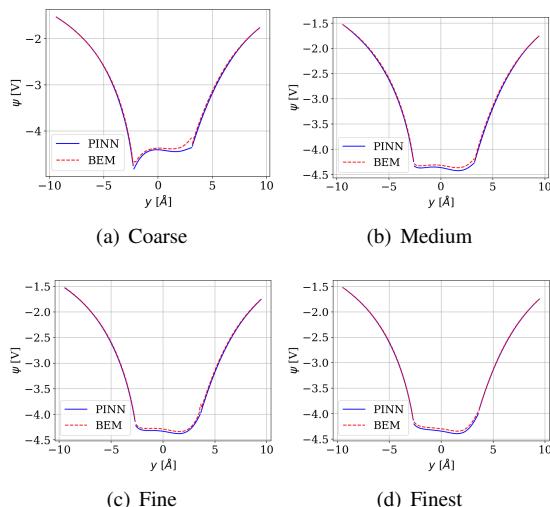


FIG. 11. Reaction potential ( $\psi$ ) for arginine with different node refinements, along the y axis. Red line: BEM solution. Blue line: PINN solution.

The surface and volume meshes used to generate the collocation points are related because the tetrahedrons in  $\Omega_m$  and  $\Omega_w$  conform to the triangles in  $\Gamma$  and  $\partial\Omega$ . However, there is no clear reason why the surface and volume collocation points should be coupled. To decouple them, we performed two extra calculations where, starting from the collocation points from the *Finest* calculation, we sample only a subset of the volume collocation points, while using all surface nodes. This way, the geometrical details of  $\Gamma$  remain constant, while decreasing the sampling size in the volume, and hence, the computational

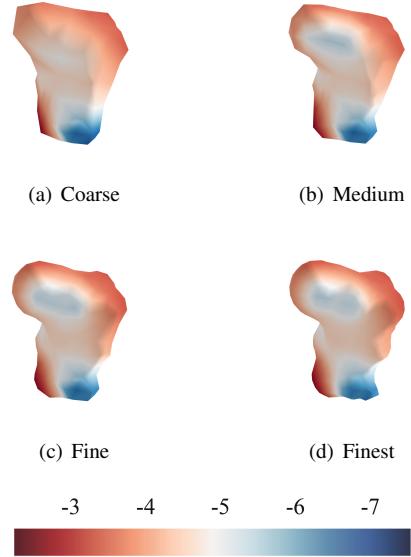


FIG. 12. Reaction potential ( $\psi_\theta^{(T)}$ , in Volts) on the molecular surface of arginine, for different collocation node density.

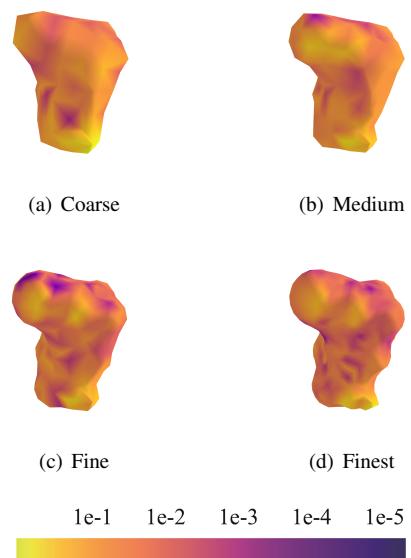


FIG. 13. Absolute difference (in Volts) in reaction potential between PINN and BEM on the molecular surface of arginine, for different collocation node density.

cost of each iteration.

Table V details the sampling size, which correspond to 30% and 60% of the volume nodes. Results are presented in Table VI at 20,000 iterations (like Table IV) and 35,000 iterations. The logic behind exploring results with more iterations is that, as only a subset of the tetrahedral volumes are considered to

generate the collocation nodes, more iterations may be necessary to correctly sample the whole space. The latter intuition is somewhat true, as the  $D_{G_{solv}}$  and  $D_\psi$  improve from 20,000 to 35,000 iterations for both *Finest 30%* and *Finest 60%*. However, there is no significant improvement in the indicators from Table VI compared to the *Finest* case in Table IV, with  $D_{G_{solv}}$  and  $D_\psi$  being in the order of  $10^{-2}$ — $10^{-3}$ , consistent with other results in this work. This demonstrates that sampling a subset of volume nodes while maintaining the surface nodes is an effective strategy towards lowering the computational cost without sacrificing accuracy.

TABLE V. Number of collocation nodes in each domain and surface, when sampling a subset of the volume nodes.

Simulation	$\Omega_m$	$\Omega_w$	SES	$\partial\Omega$
Finest 30%	2282	4589	1318	1238
Finest 60%	4494	9090	1318	1238

TABLE VI. Results for arginine with subset of volume collocation nodes.

Case	Iter.	$D_{G_{solv}}$	$D_\psi$	Train. loss	Val. loss
Finest 30%	20000	1.10E-02	8.05E-03	5.25E-06	8.10E-06
Finest 30%	35000	8.72E-03	7.78E-03	2.84E-06	1.28E-05
Finest 60%	20000	9.85E-03	9.23E-03	4.31E-06	5.27E-06
Finest 60%	35000	9.63E-03	6.71E-03	3.82E-06	1.45E-05

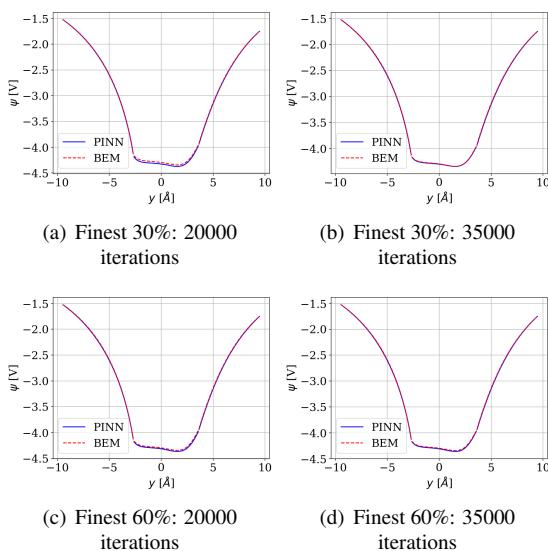


FIG. 14. Reaction potential ( $\psi$ ) for arginine along the  $y$  axis using the volumetric sub-sampling. Red line: BEM solution. Blue line: PINN solution.

## 2. Incorporating experimental measurements

One attractive feature of PINN that sets it apart from standard numerical techniques is the freedom to add loss functions with information from other sources, like different models or experimental measurements (see Eq. (14)). This is specially exciting in molecular electrostatics, as recent advances in NMR spectroscopy are capable of measuring effective electrostatic potentials ( $\phi_{ENS}$ ) around hydrogen atoms of a molecule.<sup>94</sup> This quantity is computed from a simulation as

$$\phi_{ENS} = -\frac{k_B T}{e} \ln \frac{\Gamma_{2,+}}{\Gamma_{2,-}} \quad (29)$$

where the  $\Gamma_{2,+}$  and  $\Gamma_{2,-}$  are the rate of transverse magnetization, which is:

$$\Gamma_{2,\pm} = C_0 \int_{\Omega_w} r^{-6} \exp \left\{ -\frac{\pm e \phi}{k_B T} \right\} dV, \quad (30)$$

$C_0$  being a constant that depends on NMR parameters, but is irrelevant to our case, as it is canceled out in the ratio  $\Gamma_{2,+}/\Gamma_{2,-}$  of Eq. (29).

We performed calculations on arginine using the same setup as *Finest* in Table III, this time including the following loss term:

$$L_{\phi_{ENS}}(\theta_w; S_{\Omega_w}) = \frac{1}{N_{H_{at}}} \sum_{H_{at} \in \text{atoms}} \left[ \phi_{ENS}^\theta(H_{at,i}) - \phi_{ENS}^{\text{exp}} \right]^2 \quad (31)$$

where  $\phi_{ENS}^{\text{exp}}$  is the experimental measurement,  $\phi_{ENS}^\theta$  the PINN calculation, and the sum is over the  $N_{H_{at}}$  hydrogen atoms ( $H_{at}$ ) of the molecule where the measurement is performed. In this case, we used  $\phi_{ENS}$  computed with the PBE (using BEM through PBJ<sup>19</sup>) as  $\phi_{ENS}^{\text{exp}}$ , as it was shown to be a good approximation.<sup>94</sup> To correctly compute the integral in Eq. (30), we had to extend the domain  $\Omega_w$  to  $d_{\partial\Omega} = 7 \text{ \AA}$ .

During the training process, the significant oscillations in the electrostatic potential led to large values in the exponential term of Eq. (30), causing the solution to diverge. To mitigate this issue, the exponential function was approximated using its series expansion:  $\exp(x) = 1.0 + x + x^2/2! + x^3/3! + x^4/4!$ .

The results for this setup are presented in Figs. 15 and 16, and Table VII. Fig. 15 shows that the loss function decreases with increasing number of iterations for arginine, with results in Table VII that are comparable with the *Finest* case in Table IV. Fig. 16 also shows a similar behavior to the case with-

out the experimental loss function in Fig. 11(d). Moreover, the relative differences in  $\phi_{ENS}$  between PINN and BEM for two hydrogens in the arginine structure in Table VIII are also in the  $\sim 10^{-2}$ — $10^{-3}$  range. Even though the experimental loss in Fig. 15 is noisy, it is consistently lower than the others losses, indicating that PINN incorporates the experimental  $\phi_{ENS}$  appropriately.

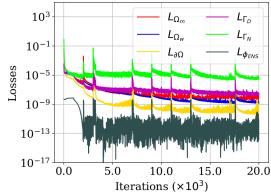


FIG. 15. Evolution of losses for arginine including Equation (31)

TABLE VII. Results for arginine considering a loss function that includes experimental measurements (Eq. (31)) after 20,000 iterations.

$\Delta G_{solv,\theta}$ kcal/mol	$D_{G_{solv}}$	$D_\psi$	Training loss	Validation loss
-134.86	8.24E-03	1.09E-02	7.88E-06	1.42E-05

#### D. Full proteins

To show the applicability of PINN in real problems, we computed the electrostatic potential and  $\Delta G_{solv}$  of the immunoglobulin-binding domain of protein G (PDB code 1pgb,<sup>95</sup> 855 atoms) and ubiquitin (PDB code 1ubq,<sup>96</sup> 1231 atoms). Following the conclusions from the results for the sphere and arginine, we considered all the features detailed in Section II D (*ie.* WA+TF+FF+SI+SO), and sampling a subset of the tetrahedral volumes to generate the collocation nodes, resulting in the parameters detailed by Table IX. The mesh settings that led to Table IX are shown in Table X.

Table XI contains the results for 1pgb and 1ubq after 40,000 iterations. The difference with a reference BEM solution (using the same surface mesh as in the creation of the collocation

TABLE VIII. Relative difference in the prediction of  $\phi_{ENS}$  for two hydrogens in arginine.

Hydrogen number	$\frac{ \phi_{ENS}^\theta - \phi_{ENS}^{exp} }{\phi_{ENS}^{exp}}$
1	9.80E-03
2	1.11E-02

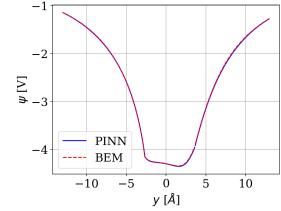


FIG. 16. Reaction potential ( $\psi$ ) for arginine along the  $y$  axis using experimental measurements.

TABLE IX. Number of collocation nodes for full proteins.

Case	$\Omega_m$	$\Omega_w$	$\Gamma$	$\partial\Omega$
1pgb	33,252	110,941	10,092	23,976
1ubq	44,751	159,199	13,064	33,122

nodes) is of the order of  $10^{-2}$  in both  $\Delta G_{solv}$  and surface potential, which is similar to arginine in Table IV. Considering these calculations run for 40,000 iterations, it is not surprising that the training and validation losses go lower than arginine, to  $10^{-6}$ , however, by looking at the evolution of  $\Delta G_{solv,\theta}$  in Figs. 17 and 20, we see it has stagnated and has reasonable results by 7,000 iterations or less.

Figs. 19 and 22 show the electrostatic potential along the  $x$  and  $y$  axis for 1pgb and 1ubq, respectively. Regardless of the high number of iterations and low difference in energy, differences in reaction potential are more evident. However, PINN is capable of reproducing the main features of the solution appropriately, like large gradient changes across the interface.

The low difference between BEM and PINN in the surface reaction potential ( $\psi^{(\Gamma)}$ ) of Table XI is further illustrated by the excellent agreement between the BEM and PINN solutions in Figs. 18 and 21. This remarkable result opens possibilities to use PINN in applications where the electrostatic potential on the surface is key, like the detection of binding pockets in drug design.

## IV. CHALLENGES AND FUTURE WORK

The results in Section III are evidence that PINN is a viable alternative to solve the linear PBE in real proteins. Although this is a promising statement, in our exploration we identified a few challenges moving towards making PINN a mainstream tool in electrostatic calculations. We hope this section will inspire researchers to address them, in order to take full advantage of neural networks in computing the electrostatic potential in molecular systems.

TABLE X. Mesh settings to generate collocation nodes for 1pgb and 1ubq.

Surf. dens. vert/ $\text{\AA}^2$		Max. vol. size $\text{\AA}^3$		sample
$\Gamma$	$\partial\Omega$	$\Omega_m$	$\Omega_w$	
1.8	1.6	0.6	2.0	70%

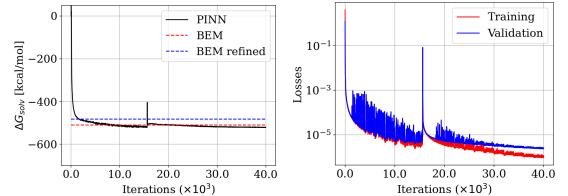
TABLE XI. Results for full proteins after 40,000 iterations.

Case	$\Delta G_{solv,\theta}$ [kcal/mol]	$D_{G_{solv}}$	$D_\psi$	Training loss	Validation loss
1pgb	-520.14	1.99E-02	3.41E-02	1.06E-06	2.50E-06
1ubq	-630.10	2.82E-02	4.69E-02	7.99E-07	2.52E-06

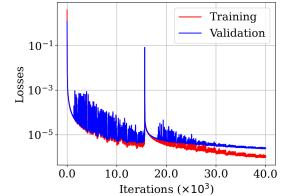
All results in Section III correspond to calculations of the linearized version of the PBE. This is a good approximation for a large family of molecules, including many proteins, however, it becomes problematic for highly charged systems, like RNA and DNA, and the nonlinear PBE in Equation (2) is required. On the other hand, PINN is a very flexible method, as the partial differential equation is easily interchangeable by modifying the loss functions  $L_{\Omega_m}$  or  $L_{\Omega_w}$  in Equation (13).

We modified our PINN code to calculate the nonlinear PBE for a DNA strand (PDB code 9ant<sup>97</sup>). The large oscillations of the electrostatic potential throughout iterations, generated large values of sinh in Equation (2), and made the solution diverge. Approximating sinh with the expansion  $\sinh x = x + x^3/3! + x^5/5!$  alleviated this issue, however, the solution showed fluctuations in the solvent region ( $\Omega_w$ ), which did not decay with iterations. Fig. 23 corresponds to the reaction potential along the  $x$  and  $y$  axis, compared with the finite difference code APBS,<sup>7</sup> showing that the PINN solution deviates from APBS in the outer region, oscillating around the correct value. Moreover, in Fig. 23, it appears that the PINN solution is unable to replicate the behavior of the finite difference solution in the molecular region. Also, the energy calculation with the non linear PBE involves a volume integral of the electrostatic potential in  $\Omega_w$ .<sup>98</sup> Due to the oscillations in the solvent region reported by Fig. 23, we were unable to compute the integral accurately, yielding nonsensical values of  $\Delta G_{solv,\theta}$ .

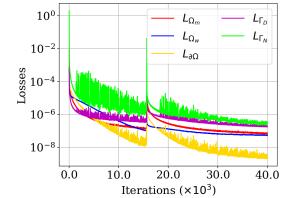
Another area of potential improvement is the optimization algorithm. In this work we used the common Adam optimizer, however, there are well-known alternatives that may converge faster, like the limited memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm.<sup>99</sup> Unfortunately the L-BFGS algorithm is not implemented in Ten-



(a) Solvation energy

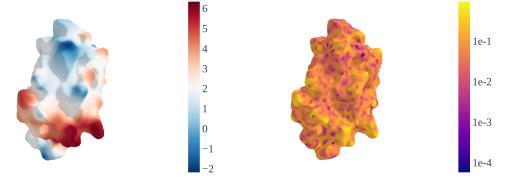


(b) Losses

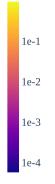


(c) Loss terms

FIG. 17. Solvation energy and losses evolution for 1pgb



(a) Reaction potential



(b) Absolute error

FIG. 18. Reaction potential ( $\psi$ ) and absolute error (in Volts) at  $\Gamma$  for 1pgb.

sorFlow. We consider that exploring other optimization algorithms and memory efficiency strategies is a line of future work that may make PINN more competitive in front of traditional numerical methods.

## V. CONCLUSIONS

This work presents a thorough investigation of PINN to solve the PBE in molecular electrostatics. Starting from a basic implementation of PINN for an interface linear PBE problem using MLP, we explored the impact of different enhancement techniques using spherical test cases with available analytical results. We conclude that the best performing neural network architecture includes layers that scale the input and output, and add random Fourier features; and also that considers a trainable activation function, and a weight balancing algorithm. We further test this architecture on realistic molecular geometries, like a single arginine, protein G (1pgb), and ubiquitin (1ubq), where we also propose a collocation node

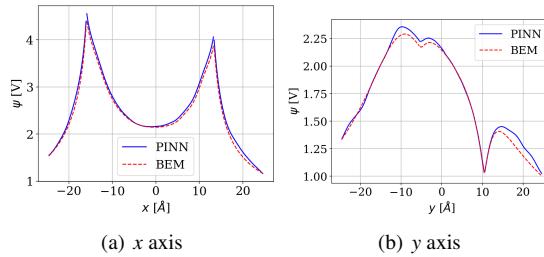


FIG. 19. Reaction potential ( $\psi$ ) along the  $x$  and  $y$  axis for 1pgb comparing PINN (blue) and BEM (red).

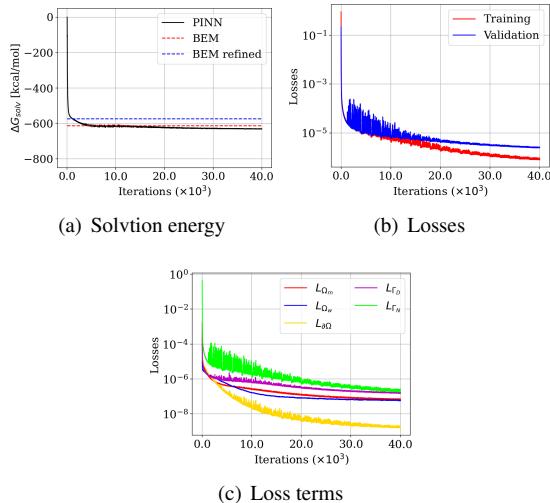


FIG. 20. Solvation energy and losses evolution for 1ubq

sampling strategy that decreases the computational cost. In all cases, PINN was capable of reproducing the details of the electrostatic potential field correctly, and the solvation energy converged to a reference value up to  $\sim 10^{-2} - 10^{-3}$ . We find that including all the features described in this work is crucial for PINN to appropriately solve the linear PBE.

We also explore the possibility to consider experimental information to our model. Using arginine as a test case, we were able to incorporate a loss function that includes NMR-measured effective electrostatic potentials near hydrogen atoms. This is an important result, as it makes PINN stand out with respect to standard numerical methods that do not have this capability.

Finally, the manuscript describes challenges encountered in our work, that may serve as a guide for coming developments to make PINN a competitive alternative to traditional numerical methods. In particular, the non-linearity of the PBE presents a difficulty, as PINN struggles to diminish oscillations in the solvent domain, affecting the energy calculation.

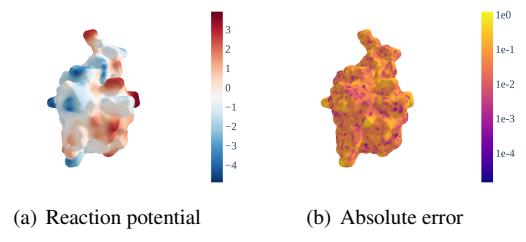


FIG. 21. Reaction potential ( $\psi$ ) and absolute error (in Volts) at  $\Gamma$  for 1ubq

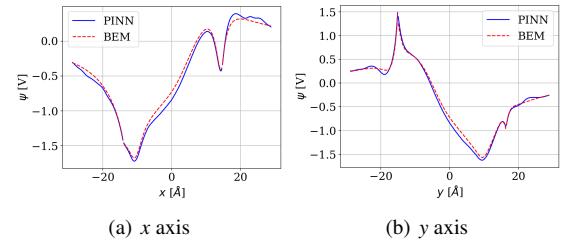


FIG. 22. Reaction potential ( $\psi$ ) along the  $x$  and  $y$  axis for 1ubq comparing PINN (blue) and BEM (red).

Also, more memory-efficient optimization algorithms are a key line of future work.

## DATA AVAILABILITY STATEMENT

All data and software required to reproduce the results of this manuscript can be found in the GitHub repository <https://github.com/MartinAchondo/XPPBE>.

## ACKNOWLEDGEMENTS

MAM thanks the support from ANID (Chile) through Beca de Magíster Nacional 22230566. CDC acknowledges the support from CCTVal through ANID PIA/APOYO AFB 220004 and Universidad Técnica Federico Santa María through from Proyectos Internos PI-LIR-23-03.

<sup>1</sup>B. Roux and T. Simonson, “Implicit solvent models,” *Biophysical chemistry* **78**, 1–20 (1999).

<sup>2</sup>S. Decherchi, M. Masetti, I. Vyalov, and W. Rocchia, “Implicit solvent methods for free energy estimation,” *European Journal of Medicinal Chemistry* **91**, 27–42 (2015).

<sup>3</sup>M. K. Gilson, A. Rashin, R. Fine, and B. Honig, “On the calculation of electrostatic interactions in proteins,” *Journal of Molecular Biology* **184**, 503–516 (1985).

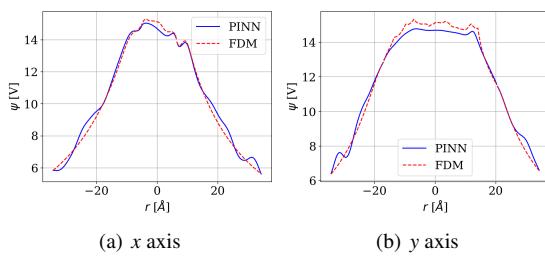


FIG. 23. Reaction potential ( $\psi$ ) along the  $x$  and  $y$  axis for a DNA strand (9ant) using an approximated nonlinear PBE. The blue line corresponds to the PINN solution and the red line is a nonlinear PBE calculation with APBS<sup>7</sup>

- <sup>4</sup>N. A. Baker, D. Sept, M. J. Holst, and J. A. McCammon, “Electrostatics of nanoystems: Application to microtubules and the ribosome,” *Proceedings of the National Academy of Sciences of the USA* **98**, 10037–10041 (2001).
- <sup>5</sup>W. Rocchia, E. Alexov, and B. Honig, “Extending the applicability of the nonlinear poisson–boltzmann equation: multiple dielectric constants and multivalent ions,” *The Journal of Physical Chemistry B* **105**, 6507–6514 (2001).
- <sup>6</sup>A. H. Boschitsch and M. O. Fenley, “A fast and robust poisson–boltzmann solver based on adaptive cartesian grids,” *Journal of Chemical Theory and Computation* **7**, 1524–1540 (2011).
- <sup>7</sup>E. Jurrus, D. Engel, K. Star, K. Monson, J. Brandi, L. E. Felberg, D. H. Brookes, L. Wilson, J. Chen, K. Liles, *et al.*, “Improvements to the apbs biomolecular solvation software suite,” *Protein Science* **27**, 112–128 (2018).
- <sup>8</sup>C. M. Cortis and R. A. Friesner, “Numerical solution of the poisson–boltzmann equation using tetrahedral finite-element meshes,” *Journal of Computational Chemistry* **18**, 1591–1608 (1997).
- <sup>9</sup>L. Chen, M. J. Holst, and J. Xu, “The finite element approximation of the nonlinear poisson–boltzmann equation,” *SIAM journal on numerical analysis* **45**, 2298–2320 (2007).
- <sup>10</sup>D. Xie and S. Zhou, “A new minimization protocol for solving nonlinear poisson–boltzmann mortar finite element equation,” *BIT Numerical Mathematics* **47**, 853–871 (2007).
- <sup>11</sup>S. D. Bond, J. H. Chaudhry, E. C. Cyr, and L. N. Olson, “A first-order system least-squares finite element method for the poisson–boltzmann equation,” *Journal of Computational Chemistry* **31**, 1625–1635 (2010).
- <sup>12</sup>P. B. Shaw, “Theory of the Poisson Green’s-function for discontinuous dielectric media with an application to protein biophysics,” *Physical Review A* **32**, 2476–2487 (1985).
- <sup>13</sup>B. J. Yoon and A. M. Lenhoff, “A boundary element method for molecular electrostatics with electrolyte effects,” *Journal of Computational Chemistry* **11**, 1080–1086 (1990).
- <sup>14</sup>A. Juffer, E. F. Botta, B. A. van Keulen, A. van der Ploeg, and H. J. Berendsen, “The electric potential of a macromolecule in a solvent: A fundamental approach,” *Journal of Computational Physics* **97**, 144–171 (1991).
- <sup>15</sup>A. H. Boschitsch, M. O. Fenley, and H.-X. Zhou, “Fast boundary element method for the linear poisson–boltzmann equation,” *The Journal of Physical Chemistry B* **106**, 2741–2754 (2002).
- <sup>16</sup>B. Lu, X. Cheng, J. Huang, and J. A. McCammon, “Order n algorithm for computation of electrostatic interactions in biomolecular systems,” *Pro-*

- ceedings of the National Academy of Sciences
- <sup>17</sup>W. Geng and R. Krasny, “A treecode-accelerated boundary integral poisson–boltzmann solver for electrostatics of solvated biomolecules,” *Journal of Computational Physics* **247**, 62–78 (2013).
- <sup>18</sup>C. D. Cooper, J. P. Bardhan, and L. A. Barba, “A biomolecular electrostatics solver using Python, GPUs and boundary elements that can handle solvent-filled cavities and Stern layers.” *Computer Physics Communications* **185**, 720–729 (2014).
- <sup>19</sup>S. D. Search, C. D. Cooper, and E. Van’t Wout, “Towards optimal boundary integral formulations of the poisson–boltzmann equation for molecular electrostatics,” *Journal of Computational Chemistry* **43**, 674–691 (2022).
- <sup>20</sup>L. E. Felberg, D. H. Brookes, E.-H. Yap, E. Jurrus, N. A. Baker, and T. Head-Gordon, “Pb-am: An open-source, fully analytical linear poisson–boltzmann solver,” *Journal of computational chemistry* **38**, 1275–1282 (2017).
- <sup>21</sup>S. V. Siryk and W. Rocchia, “Arbitrary-shape dielectric particles interacting in the linearized poisson–boltzmann framework: An analytical treatment,” *The Journal of Physical Chemistry B* **126**, 10400–10426 (2022).
- <sup>22</sup>A. Jha, M. Nottoli, A. Mikhalev, C. Quan, and B. Stamm, “Linear scaling computation of forces for the domain-decomposition linear poisson–boltzmann method,” *The Journal of Chemical Physics* **158** (2023).
- <sup>23</sup>A. Jha and B. Stamm, “Domain decomposition method for poisson–boltzmann equations based on solvent excluded surface,” *arXiv preprint arXiv:2309.06862* (2023).
- <sup>24</sup>A. H. Boschitsch and M. O. Fenley, “Hybrid boundary element and finite difference method for solving the nonlinear poisson–boltzmann equation,” *Journal of Computational Chemistry* **25**, 935–955 (2004).
- <sup>25</sup>J. Ying and D. Xie, “A hybrid solver of size modified poisson–boltzmann equation by domain decomposition, finite element, and finite difference,” *Applied Mathematical Modelling* **58**, 166–180 (2018).
- <sup>26</sup>M. Bosy, M. W. Scroggs, T. Betcke, E. Burman, and C. D. Cooper, “Coupling finite and boundary element methods to solve the poisson–boltzmann equation for electrostatics in molecular solvation,” *Journal of Computational Chemistry* **45**, 787–797 (2024).
- <sup>27</sup>M. Kapteyn, D. Knezevic, D. Huynh, M. Tran, and K. Willcox, “Data-driven physics-based digital twins via a library of component-based reduced-order models,” *International Journal for Numerical Methods in Engineering* **123**, 2986–3003 (2022).
- <sup>28</sup>M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, and Prabhat, “Deep learning and process understanding for data-driven earth system science,” *Nature* **566**, 195–204 (2019).
- <sup>29</sup>D. Calvetti and E. Somersalo, *Introduction to Bayesian Scientific Computing* (Springer New York, 2007).
- <sup>30</sup>P. Benner, S. Gugercin, and K. Willcox, “A survey of projection-based model reduction methods for parametric dynamical systems,” *SIAM Review* **57**, 483–531 (2015).
- <sup>31</sup>G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, “Physics-informed machine learning,” *Nature Reviews Physics* **3**, 422–440 (2021).
- <sup>32</sup>M. G. Dissanayake and N. Phan-Thien, “Neural-network-based approximations for solving partial differential equations,” *communications in Numerical Methods in Engineering* **10**, 195–201 (1994).
- <sup>33</sup>M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse prob-

- lems involving nonlinear partial differential equations," *Journal of Computational physics* **378**, 686–707 (2019).
- <sup>34</sup>S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics-informed neural networks: Where we are and what's next," *Journal of Scientific Computing* **92**, 88 (2022).
- <sup>35</sup>S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (pinns) for fluid mechanics: a review," *Acta Mechanica Sinica* **37**, 1727–1738 (2021).
- <sup>36</sup>S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis, "Physics-Informed Neural Networks for Heat Transfer Problems," *Journal of Heat Transfer* **143**, 060801 (2021).
- <sup>37</sup>M. Baldan, P. Di Barba, and D. A. Lowther, "Physics-informed neural networks for inverse electromagnetic problems," *IEEE Transactions on Magnetics* **59**, 1–5 (2023).
- <sup>38</sup>J. D. Schmid, P. Bauerschmidt, C. Gurbuz, M. Eser, and S. Marburg, "Physics-informed neural networks for acoustic boundary admittance estimation," *Mechanical Systems and Signal Processing* **215**, 111405 (2024).
- <sup>39</sup>C. Eldred, F. Gay-Balmaz, S. Huraka, and V. Putkaradze, "Lie–poisson neural networks (lpnets): Data-based computing of hamiltonian systems with symmetries," *Neural Networks* **173**, 106162 (2024).
- <sup>40</sup>B. Yu *et al.*, "The deep ritz method: a deep learning-based numerical algorithm for solving variational problems," *Communications in Mathematics and Statistics* **6**, 1–12 (2018).
- <sup>41</sup>E. Kharazmi, Z. Zhang, and G. E. Karniadakis, "Variational physics-informed neural networks for solving partial differential equations," *arXiv preprint arXiv:1912.00873* (2019).
- <sup>42</sup>G. Lin, P. Hu, F. Chen, X. Chen, J. Chen, J. Wang, and Z. Shi, "Binet: Learning to solve partial differential equations with boundary integral networks," *arXiv preprint arXiv:2110.00352* (2021).
- <sup>43</sup>J. Sun, Y. Liu, Y. Wang, Z. Yao, and X. Zheng, "Binn: A deep learning approach for computational mechanics problems based on boundary integral equations," *Computer Methods in Applied Mechanics and Engineering* **410**, 116012 (2023).
- <sup>44</sup>A. D. Jagtap and G. E. Karniadakis, "Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations," *Communications in Computational Physics* **28** (2020).
- <sup>45</sup>A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," *Computer Methods in Applied Mechanics and Engineering* **365**, 113028 (2020).
- <sup>46</sup>V. Dwivedi, N. Parashar, and B. Srinivasan, "Distributed physics informed neural network for data-efficient solution to partial differential equations," *arXiv preprint arXiv:1907.08967* (2019).
- <sup>47</sup>W. Li, X. Xiang, and Y. Xu, "Deep domain decomposition method: Elliptic problems," in *Mathematical and Scientific Machine Learning* (PMLR, 2020) pp. 269–286.
- <sup>48</sup>C. He, X. Hu, and L. Mu, "A mesh-free method using piecewise deep neural network for elliptic interface problems," *Journal of Computational and Applied Mathematics* **412**, 114358 (2022).
- <sup>49</sup>S. Wu and B. Lu, "INN: Interfaced neural networks as an accessible meshless approach for solving interface PDE problems," *Journal of Computational Physics* **470**, 111588 (2022).
- <sup>50</sup>J. Ying, J. Liu, J. Chen, S. Cao, M. Hou, and Y. Chen, "Multi-scale fusion network: A new deep learning structure for elliptic interface problems," *Applied Mathematical Modelling* **114**, 252–269 (2023).
- <sup>51</sup>Y.-H. Tseng, T.-S. Lin, W.-F. Hu, and M.-C. Lai, "A cusp-capturing PINN for elliptic interface problems," *Journal of Computational Physics* **491**, 112359 (2023).
- <sup>52</sup>X. Jiang, Z. Wang, W. Bao, and Y. Xu, "Generalization of PINNs for elliptic interface problems," *Applied Mathematics Letters* , 109175 (2024).
- <sup>53</sup>A. K. Sarma, S. Roy, C. Annavarapu, P. Roy, and S. Jagannathan, "Interface pinns (i-pinns): A physics-informed neural networks framework for interface problems," *Computer Methods in Applied Mechanics and Engineering* **429**, 117135 (2024).
- <sup>54</sup>S. Wu, A. Zhu, Y. Tang, and B. Lu, "Convergence of physics-informed neural networks applied to linear second-order elliptic interface problems," *Communications in Computational Physics* **33**, 596–627 (2023).
- <sup>55</sup>Z. Liu, W. Cai, and Z.-Q. John Xu, "Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains," *Communications in Computational Physics* **28**, 1970–2001 (2020).
- <sup>56</sup>S. Wu, A. Zhu, Y. Tang, and B. Lu, "Solving parametric elliptic interface problems via interfaced operator network," *Journal of Computational Physics* **514**, 113217 (2024).
- <sup>57</sup>J.-A. Désidéri, "Multiple-gradient descent algorithm (mgda) for multi-objective optimization," *Comptes Rendus Mathematique* **350**, 313–318 (2012).
- <sup>58</sup><https://www.tensorflow.org/>.
- <sup>59</sup><https://pytorch.org/>.
- <sup>60</sup>D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980* (2014).
- <sup>61</sup>D. C. Whitley, "Van der waals surface graphs and molecular shape," *Journal of mathematical chemistry* **23**, 377–397 (1998).
- <sup>62</sup>M. L. Connolly, "Molecular surface triangulation," *Journal of Applied Crystallography* **18**, 499–505 (1985).
- <sup>63</sup>J. Chen, Y. Xu, X. Yang, Z. Cang, W. Geng, and G.-W. Wei, "Poisson-boltzmann-based machine learning model for electrostatic analysis," *Biochemical Journal* (2024).
- <sup>64</sup>P. B. Reis, D.-A. Clevert, and M. Machuqueiro, "Pypka server: online p k a predictions and biomolecular structure preparation with precomputed data from pdb and alphafold db," *Nucleic Acids Research* , gkae255 (2024).
- <sup>65</sup>M. Achondo, "XPPBE: PINNs for PBE," <https://github.com/MartinAchondo/XPPBE> (2024).
- <sup>66</sup>P.-Y. Chuang and L. A. Barba, "Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration," *arXiv preprint arXiv:2205.14249* (2022).
- <sup>67</sup>T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb, "Can physics-informed neural networks beat the finite element method?" *IMA Journal of Applied Mathematics* , hxae011 (2024).
- <sup>68</sup>T. Kurth, S. Subramanian, P. Harrington, J. Pathak, M. Mardani, D. Hall, A. Miele, K. Kashinath, and A. Anandkumar, "Fourcastnet: Accelerating global high-resolution weather forecasting using adaptive fourier neural operators," in *Proceedings of the Platform for Advanced Scientific Computing Conference, PASCA '23* (Association for Computing Machinery, New York, NY, USA, 2023).
- <sup>69</sup>K. Bi, L. Xie, H. Zhang, X. Chen, X. Gu, and Q. Tian, "Accurate medium-range global weather forecasting with 3d neural networks," *Nature* **619**,

- 533–538 (2023).
- <sup>70</sup>A. D. Jagtap, Z. Mao, N. Adams, and G. E. Karniadakis, “Physics-informed neural networks for inverse problems in supersonic flows,” *Journal of Computational Physics* **466**, 111402 (2022).
- <sup>71</sup>B. Lee and F. Richards, “The interpretation of protein structures: Estimation of static accessibility,” *Journal of Molecular Biology* **55**, 379–IN4 (1971).
- <sup>72</sup>Z. Yu, M. P. Jacobson, and R. A. Friesner, “What role do surfaces play in gb models? a new-generation of surface-generalized born model based on a novel gaussian surface for biomolecules,” *Journal of computational chemistry* **27**, 72–89 (2006).
- <sup>73</sup>Z. Zhou, P. Payne, M. Vasquez, N. Kuhn, and M. Levitt, “Finite-difference solution of the Poisson-Boltzmann equation: Complete elimination of self-energy,” *J. Comput. Chem.* **17**, 1344–1351 (1996).
- <sup>74</sup>M. Holst, J. A. Mccammon, Z. Yu, Y. Zhou, and Y. Zhu, “Adaptive finite element modeling techniques for the poisson-boltzmann equation,” *Communications in computational physics* **11**, 179–214 (2012).
- <sup>75</sup>A. Lee, W. Geng, and S. Zhao, “Regularization methods for the poisson-boltzmann equation: comparison and accuracy recovery,” *Journal of Computational Physics* **426**, 109958 (2021).
- <sup>76</sup>J. Che, J. Dzubiella, B. Li, and J. A. McCammon, “Electrostatic free energy and its variations in implicit solvent models,” *The Journal of Physical Chemistry B* **112**, 3058–3069 (2008).
- <sup>77</sup>I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning* (MIT press, 2016).
- <sup>78</sup>A. L. Caterini and D. E. Chang, *Deep neural networks in a mathematical framework* (Springer, 2018).
- <sup>79</sup>A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, “Automatic differentiation in machine learning: a survey,” *Journal of machine learning research* **18**, 1–43 (2018).
- <sup>80</sup>Y. Shin, J. Darbon, and G. E. Karniadakis, “On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes,” arXiv preprint arXiv:2004.01806 (2020).
- <sup>81</sup>M. A. Nabian, R. J. Gladstone, and H. Meidani, “Efficient training of physics-informed neural networks via importance sampling,” *Computer-Aided Civil and Infrastructure Engineering* **36**, 962–977 (2021).
- <sup>82</sup>C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, “A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks,” *Computer Methods in Applied Mechanics and Engineering* **403**, 115671 (2023).
- <sup>83</sup>S. Wang, S. Sankaran, H. Wang, and P. Perdikaris, “An Expert’s Guide to Training Physics-informed Neural Networks,” arXiv preprint arXiv:2308.08468 (2023).
- <sup>84</sup>M. J. Holst *et al.*, “The Poisson-Boltzmann equation: Analysis and multi-level numerical solution,” *Applied Mathematics and CRPC*, California Institute of Technology (1994).
- <sup>85</sup>M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, “Fourier features let networks learn high frequency functions in low dimensional domains,” *Advances in neural information processing systems* **33**, 7537–7547 (2020).
- <sup>86</sup>A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, “Adaptive activation functions accelerate convergence in deep and physics-informed neural networks,” *Journal of Computational Physics* **404**, 109136 (2020).
- <sup>87</sup>M. F. Sanner, A. J. Olson, and J.-C. Spehner, “Reduced surface: an efficient way to compute molecular surfaces,” *Biopolymers* **38**, 305–320 (1996).
- <sup>88</sup>S. Decherchi and W. Rocchia, “A general and robust ray-casting-based algorithm for triangulating surfaces at the nanoscale,” *PLOS one* **8**, e59744 (2013).
- <sup>89</sup>S. Hang, “Tetgen, a delaunay-based quality tetrahedral mesh generator,” *ACM Trans. Math. Softw.* **41**, 11 (2015).
- <sup>90</sup>C. T. Lee, J. G. Laughlin, J. B. Moody, R. E. Amaro, J. A. McCammon, M. Holst, and P. Rangamani, “An open-source mesh generation platform for biophysical modeling using realistic cellular geometries,” *Biophysical Journal* **118**, 1003–1008 (2020).
- <sup>91</sup>J. G. Kirkwood, “Theory of solutions of molecules containing widely separated charges with special application to zwitterions,” *The Journal of Chemical Physics* **2**, 351–361 (1934).
- <sup>92</sup><https://github.com/bem4solvation/pbj>.
- <sup>93</sup><https://www.rcsb.org/ligand/ARG>.
- <sup>94</sup>B. Yu, C. C. Pletka, B. M. Pettitt, and J. Iwahara, “De novo determination of near-surface electrostatic potentials by nmr,” *Proceedings of the National Academy of Sciences* **118**, e2104020118 (2021).
- <sup>95</sup>T. Gallagher, P. Alexander, P. Bryan, and G. L. Gilliland, “Two crystal structures of the b1 immunoglobulin-binding domain of streptococcal protein g and comparison with nmr,” *Biochemistry* **33**, 4721–4729 (1994).
- <sup>96</sup>S. Vijay-Kumar, C. E. Bugg, and W. J. Cook, “Structure of ubiquitin refined at 1.8 åresolution,” *Journal of molecular biology* **194**, 531–544 (1987).
- <sup>97</sup>E. Fraenkel and C. O. Pabo, “Comparison of x-ray and nmr structures for the antennapedia homeodomain–dna complex,” *Nature structural biology* **5**, 692–697 (1998).
- <sup>98</sup>D. Sitkoff, K. A. Sharp, and B. Honig, “Accurate calculation of hydration free energies using macroscopic solvent models,” *The Journal of Physical Chemistry* **98**, 1978–1988 (1994).
- <sup>99</sup>D. C. Liu and J. Nocedal, “On the limited memory BFGS method for large scale optimization,” *Mathematical programming* **45**, 503–528 (1989).